REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL OR	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Including Security Classification)
An Intra-Theater Transportation System Simulation to Assist Logisticians in Transportation Resource Planning and Implementation

12 PERSONAL AUTHOR(S)
JUDY, James M.

13 TYPE OF REPORT	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day)	15. Page Count
Master's thesis		1992, SEPTEMBER	164

16. SUPPLEMENTAL NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	MODSIM, Object-Oriented Programming, Simulations, Transportation System, Intra-Theater Operations

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Transportation resource planning and implementation within a theater of operations have always been challenging for logisticians. This was especially true during Operation Desert Storm, where new lessons were learned because of a scenario different than any other experienced. What was needed was a transportation asset-focused model that would allow logisticians to plan more effectively for current and future transportation system requirements. The focus of this thesis is the development of the Intra-Theater Transportation System Simulation (ITTSS). ITTSS is an object-oriented simulation model, which was developed to simulate a complete transportation system where units consume supplies, supply points resupply, and assets deliver the supplies. ITTSS can also be used to schedule specific missions of moving cargo from one location to another. Both modes can be run separately or together. A variety of input parameters concerning supply points, motorpool, maintenance facilities, fuel points, convoys and the operations performed can be adjusted to fit any specific scenario. The measures of performance produced by the model include the daily amount of cargo moved, time required to move cargo to a certain location, and the availability and utilization rates of vehicles. ITTSS is designed to run on a personal computer, using the PC-OS/2 version of MODSIM II, the OS/2 1.2 operating system, and Microsoft C 5.0.

20 DISTRIBUTION/AVAILABILTIY OF ABSTRACT	21a. REPORT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL William J. Caldwell	22b. TELEPHONE (Include Area Code) (408)646-3452	22c. OFFICE SYMBOL OR/Cw

DD Form 1473, JUN 86 — Previous editions are obsolete. S/N 0102-LF-014-6603

Approved for public release; distribution is unlimited.

AN INTRA-THEATER TRANSPORTATION SYSTEM SIMULATION TO ASSIST LOGISTICIANS IN TRANSPORTATION RESOURCE PLANNING AND IMPLEMENTATION

by

James M. Judy
Captain, United States Army
B.S., United States Military Academy, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

ABSTRACT

Transportation resource planning and implementation within a theater of operations have always been challenging for logisticians. This was especially true during Operation Desert Storm, where new lessons were learned because of a scenario different than any other experienced. What was needed was a transportation asset-focused model that would allow logisticians to plan more effectively for current and future transportation system requirements. The focus of this thesis is the development of the Intra-Theater Transportation System Simulation (ITTSS). ITTSS is an object-oriented simulation model, which was developed to simulate a complete transportation system where units consume supplies, supply points resupply, and assets deliver the supplies. ITTSS can also be used to schedule specific missions of moving cargo from one location to another. Both modes can be run separately or together. A variety of input parameters concerning supply points, motorpools, maintenance facilities, fuelpoints, convoys and the operations performed can be adjusted to fit any specific scenario. The measures of performance produced by the model include the daily amount of cargo moved, time required to move cargo to a certain location, and the availability and utilization rates of vehicles. ITTSS is designed to run on a personal computer, using the PC-OS/2 version of MODSIM II, the OS/2 1.2 operating system, and Microsoft C 5.0.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The author wishes to express his thanks and gratitude to the following individuals, without whom this project could not have been completed.

- My fiancee, Lorrie, for her long distance support of my efforts.
- My thesis advisor, LTC William Caldwell for believing in my proposal and standing by me all the way.
- Dr. David Horner, Waterways Experiment Station, Vicksburg, Mississippi, for providing vehicle travel rate data.
- Mr. Russ Farrell, U.S. Materiel Systems Analysis Activity, Aberdeen Proving Grounds, Maryland for all the reliability data.
- Captain Bernard Mimms, USMC, for MODSIM II programming support, being one of only a handfull who used the PC-OS\2 version.

I. INTRODUCTION

A. BACKGROUND

> Victory is the beautiful, bright-coloured flower. Transport is the stem without which it could never have blossomed. Sir Winston S. Churchill [Ref. 1: p. 202]

Ever since armies have gone to war, logistical support has played an important role in the outcome of battles and the strategy of war. However, managing the transport of supplies and the transportation assets within the transportation system has been far from easy. Napoleon was quoted as saying

> "that logistics make up as much as nine tenths of the business of war, and that the mathematical problems involved in calculating the movements and supply of armies are not unworthy of a Leibnitz or a Newton." [Ref. 2: p. 231]

During Operation Desert Storm, new lessons were learned in logistics because of a scenario different than any other experienced. Support requirements for the men and equipment that were deployed increased and changed dramatically. The successful accomplishments in meeting transportation requirements within the theater of operations in Saudi Arabia were instrumental in our success in the campaign.

However, the present configuration of transportation units did not support the transportation requirements. Hundreds of vehicles, especially heavy transporters, were leased from other countries to correct this deficiency. If the Army logisticians had a realistic simulation model that was focused on transportation assets, it would have assisted them in planning more effectively for current and future transportation system

requirements. Such a model would allow the same scenarios to be used to test different transportation unit organizations and future transportation asset prototypes to optimize the best mix of assets and the organization of units within the theater.

1. Operation Desert Storm

Operation Desert Storm offered enormous challenges to logisticians in the movement of vehicles and cargo within the theater of operations. They were extremely successful in meeting every transportation obstacle they faced. One obstacle was the movement of two corps before the major offensive of the war.

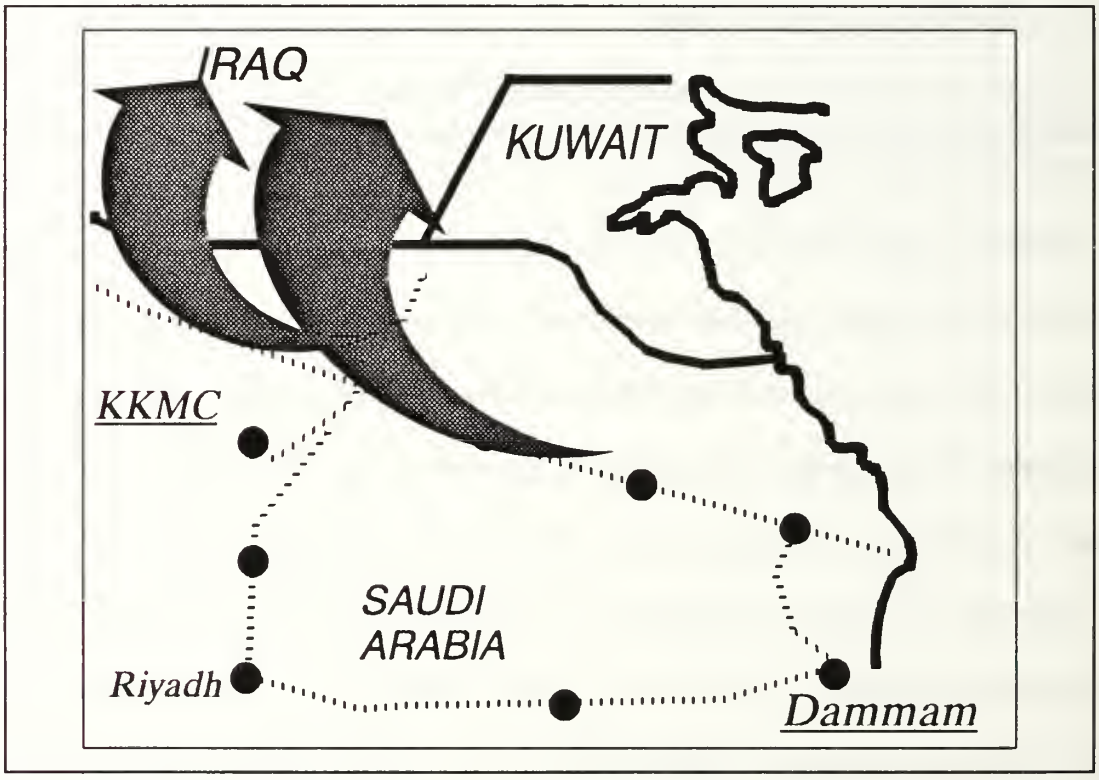

Figure 1. Corps Movements

In preparation for this offensive, the theater Commander-In-Chief (CINC) tasked his logisticians to develop a plan to move all of the VII Corps' and XVIII Airborne Corps' equipment within a two week timeframe. The logisticians did not have a model to provide them with reasonable alternatives. "After many grueling days and nights of analyzing every movement concept imaginable, [it was] concluded that the move could not be completed in 14 days" [Ref. 3: p. 8]. At least three weeks were needed because of the enormous amount of equipment involved and the limited available assets to move them.

Table 1. CORPS EQUIPMENT MOVED BEFORE OFFENSIVE

Assets Available	Loads to Move
280 Heavy Equipment Transporters	535
280 Lowboys (Equipment Transporters)	1793
500 Flatbeds (Cargo Trailers)	2815

This move was so critical to the start of the offensive that the CINC "required the SUPCOM [Support Command] CG [Commanding General] to guarantee he complete the movement in 21 days, and required the SUPCOM to sign a document attesting to the same" [Ref. 3: p. 9]. The result was that "exactly 21 days after the start date, the greatest battlefield movement in history was complete" [Ref. 3: p. 10].

If the logisticians in Operation Desert Storm had a transportation-asset-focused simulation model, it could have assisted them in estimating the time needed to move the corps equipment using their available assets. Further study of the Operation Desert Storm scenario with different numbers and kinds of assets would be possible using

this model to determine what transportation unit configurations would best suit another Operation Desert Storm type war. Using future prototypes in the same scenario could also show if having better performing or more reliable equipment could have made a difference in the amount of cargo moved daily.

2. New Transportation Corps Doctrine

The advantages in having the ability to move an entire Heavy Maneuver Force (HMF) in one lift were apparent during Operation Desert Storm. Even though this one lift concept was not used in Saudi Arabia because of the lack of heavy equipment transporters (HET), the ability to analyze the movement of such a HMF in the Operation Desert Storm scenario using HETs, would assist logisticians in planning for and implementing their HET resources for future operations in a desert scenario. Such analysis could also determine whether or not the same unit configuration would succeed in fulfilling transportation requirements within a different theater of operations. Instead of having a unit's tracked vehicles road-march several hundred miles, HETs can be used to relocate the unit. This new doctrine results in substantial fuel savings and, more importantly, a more rested and better prepared maneuver force [Ref. 4: p. 42]. Both vehicles and crews have less wear and tear. The Transportation Corps is developing the new doctrine for HET companies that can carry a battalion size HMF. Four such companies would be able to provide a single-lift capacity for a brigade size HMF. A simulation model that could focus on the HETs could test this configuration in a variety of potential scenarios including the recent Operation Desert Storm Scenario. [Ref. 4: p. 42]

Every major conflict of war produces new transportation doctrine based on the transportation system in existence, to include the modes of transportation assets available, the network of vehicles, trains, aircraft and vessels and the availability of various assets. A European scenario dictates the use of the existing rail lines. Saudi Arabia favored using HETs to relocate tracked vehicles. The next location for a major conflict is unknown, but having a transportation-asset-focused simulation model, could test what kind of transportation unit configurations would be needed and how many and what kind of assets are required to perform the mission.

B. MODEL DISCUSSION

The purpose of this thesis is to describe the development of an objected-oriented simulation model, called the Intra-Theater Transportation System Simulation (ITTSS). Currently, there are no models available that focus on individual asset performance and reliability for an intra-theater scenario. Many models use flat rates for vehicle which as resulted in standard transportation doctrine using using 83% for long hauls and 75% for short hauls for truck availability. [Ref. 5: p. 3-22] Many existing large-scale combat models aggregate transportation assets and therefore lose the ability to analyze the individual performance of these assets. A detailed simulation would assist logisticians greatly in determining transportation unit organizations to meet the requirements in a particular theater of operations. A model that could also simulate a transportation system with a variety of realistic inputs and parameters that could be varied depending on a particular scenario would further contribute to logistics analysis. The Intra-Theater

Transportation System Simulation was developed with these needs in mind. With a focus on transportation assets and flexibility to model a variety of scenarios, ITTSS will assist logisticians in transportation resource planning and implementation within an intra-theater of operations.

The recent advent of object-oriented simulation programming makes developing such a simulation model more attractive and useful than past simulation models. An object-oriented language like MODSIM II™, allows the modeling assets and cargo as actual objects that can move around and be loaded within the transportation system. Nodes can be treated as objects and have supply, motorpool, maintenance, and fuelpoint objects associated with them.

The result is a simulation model that the user can understand more clearly, and that simulates more realistically the way the elements of a transportation system actually work. Units consume supplies, supply points reorder supplies from their supply source, and vehicles load and transport these supplies. All are objects that actually perform these functions.

MODSIM II™, being modular, makes adaptation to various scenarios and unit organizations very easy. ITTSS focuses on vehicle assets, but with relative ease, can be adapted to utilize aircraft, trains, or inland waterway vessels.

Being able to focus on the movement of cargo and transportation assets in an object-oriented view gives a clearer understanding of the movement of assets and cargo. Specific assets and cargo can be tracked individually within the transportation network. Assets can have individual performance and reliability characteristics and methods that

let them perform realistic functions. Cargo can be individual items that have dimensions and be can traced within the theater of operations.

ITTSS simulates an intra-theater transportation system to include daily supply and transportation operations and specific movement of cargo from one location to another. The focus is on assets and their performance within a theater of operations.

C. ORGANIZATION OF THESIS

Chapter I has addressed the background and motivation for an intra-theater simulation model. In Chapter II the methodology in developing ITTSS, the sources for the model's data, and the defining of the transportation network is discussed. Assumptions, measures of performance, input parameters, reports and a complete description of the capabilities of ITTSS is found in Chapter III. Chapter IV provides an example comparing the new Transportation Corps doctrine with doctrine used during Operation Desert Storm and demonstrates the capabilities of the model. Concluding remarks and recommendations for future enhancements are discussed in Chapter V.

II. METHODOLOGY AND DATA

A. INTRODUCTION

ITTSS is written in an object-oriented computer language, MODSIM II™. This allows the physical entities of ITTSS's transportation system to be treated as objects. Objects have methods that allow them to perform functions and fields that describe them. MODSIM II™ also includes many convenient procedures and functions that are built in and eliminates the need to create computer code for many routine functions. This chapter discusses the sources of data for the model and describes the objects that define the basic transportation network of ITTSS.

B. DATA

1. Published Data

All performance data is taken from technical manuals or a respected source like "Jane's Military Vehicles and Logistics". [Ref. 6.] This data includes fuel consumption rates, fuel capacities, vehicle length and weights, and cargo dimensions.

2. Vehicle Speeds

Rates of travel for the selected vehicles in ITTSS are obtained from the Waterways Experiment Station (WES) in Vicksburg, Mississippi. WES does an enormous amount of mobility testing on almost every vehicle in the Army's inventory. They have provided the average rate of speed for a particular type and model of vehicle

traveling over a road with a particular type of road surface and terrain. These speeds are used when determining how long it takes a vehicle to travel over a road in the network.

3. Reliability Data

For each vehicle used in ITTSS, reliability data was obtained from the U.S. Army Material Systems Analysis Activity (AMSAA), located at Aberdeen Proving Grounds, Maryland. AMSAA does extensive testing on all Army vehicles and provided the following reliability measures.

a. Mean Miles Between Operational Mission Failure

This is the average rate of occurrence of a mission failure as a result of any cause. Mission failure is defined as not being able to complete a mission in an acceptable manner. Vehicles that fail are not able to continue with a convoy or complete the mission unless repaired. [Ref. 7: p. 1]

b. Maintenance Manhours Per Mile

This reliability factor is the average number of maintenance manhours that is needed per mile traveled by the vehicle between operational mission failures. This includes the actual wrench turning time needed to repair the vehicle. More miles between operational mission failures imply more time required to repair the asset. [Ref. 7: p. 1]

c. *Administrative and Logistics Downtime*

This is a flat rate that provides a measure of how long a vehicle is down for each operational mission failure, excluding actual maintenance wrench turning time. This includes time needed for recovery of assets, waiting for parts, tools, or mechanics. [Ref. 7: p. 1]

C. NETWORK DEFINITION

The basic transportation network of ITTSS consists of assets, nodes, links, and cargo. They are all objects that have fields and methods that distinguish and describe them.

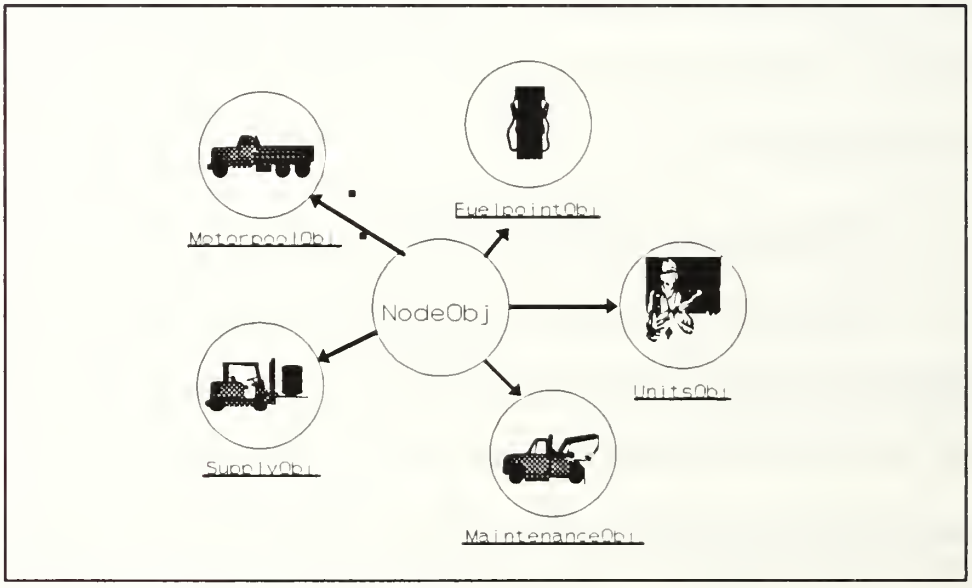

Figure 2. Objects in the Network

1. Assets

The only assets in this simulation are vehicles. Vehicles consume fuel, load cargo, and travel from one node to another. They travel in convoys along routes predetermined by the user. Vehicle objects in this simulation include cargo trucks, tractors trailers, lowboys, heavy equipment transporters and wreckers. Other types of assets like aircraft, trains, watercraft or other types of vehicles can easily be incorporated into the model with minimal code change.

2. Nodes

Nodes are objects and can have any combination of the objects listed below. See Figure 2.

a. Unit Objects

These are the consumers of the simulation model. They consume supplies from their supply point at the end of each day or at a time designated by the user. Units can be given dates to start and end consuming.

b. Supply Objects

These objects conduct supply operations at a selected time each day. Stock levels are checked and resupplies are ordered from its higher supply source object. Supplies are received from vehicles that arrive at the supply point. Supply records are used to account for items brought in and taken out of inventory. Supply objects have a certain number of loading and receiving points that serve convoys. Material Handling

Equipment (MHE) is available at each point and available MHE is used to determine how many vehicles of a convoy can be loaded or unloaded at the same time.

c. Motorpool Objects

All vehicles belong to a specific motorpool object. Motorpools schedule missions using available vehicles and keep records on availability and utilization. The matching of cargo to vehicles and loading of vehicles are conducted by motorpools. Convoys are formed and prepared for travel.

d. Maintenance Objects

Repair of vehicles occur in maintenance objects. Vehicles that breakdown are logged into the maintenance facilities, repaired and returned to their motorpools.

e. Fuelpoint Objects

These objects refuel vehicles. Fuel levels are maintained at each fuelpoint to show fuel consumption and demand. Vehicles do not run out of fuel, as a reserve is carried by the convoy in case the distance to the next fuel point is beyond the range of the convoy. Records will indicate those links in a route that vehicles would need reserve fuel. The user may then want to establish another fuelpoint.

3. Links

Links are distinguished by an origin, destination and distance in miles. Each link is characterized by a particular road surface of either concrete, bituminous (asphalt),

gravel, or dirt and a type terrain that is either flat, rolling hills, hills with curves, or mountainous.

Links are collected together to form routes and can be a part of any number of routes. More than one route list can be defined for the network to have several ground routes or air, train, or inland waterway routes.

4. Cargo

These objects are supply items in pounds or special items with dimensions in inches and pounds. Cargo objects must have a mode of resupply established. Cargo is moved by cargo trucks, trailers, lowboys, heavy equipment transporters or wreckers. Cargo can be separable or nonseparable. If separable, cargo is aggregated and separated by pounds. Classes of supply are separable in ITTSS. These cargo objects can be separated into smaller objects, so each cargo object will be a separate entity no matter how small. Tanks and other major items of interest are nonseparable and cannot be separated into smaller items. They are loaded by weight and length, and can be easily tracked throughout the network.

Cargo can be prioritized by the user. The higher the priority given a cargo, the higher priority a convoy has clearing through an intersection and offloading cargo at a supply receiving point. All cargo is accounted for by supply records and kept in supply point inventories.

III. MODEL DESCRIPTION

A. ASSUMPTIONS AND MODEL LIMITATIONS

1. Assets

In ITTSS, all assets are vehicles. Tractors are aggregated with trailers and tankers to keep the model manageable. Reliability data is available on trailers, but it is assumed in ITTSS that the determining factor of whether or not a prime mover and trailer are mission capable is the operational status of the prime mover. A trailer can usually be moved regardless of its operational status, but without its prime mover it cannot be moved at all. It is assumed that there would be no difference in the performance of the assets if you separate tractor from trailers or keep them together.

2. Combat Losses

Combat losses are not modelled in ITTSS. Future embellishments to the model could include such things as random ambushes that impact on the performance of assets and the movement of cargo and that partially or fully disable a link or node.

3. Links

Each individual link is uniform in terrain and road surface. Instead of updating a traversed link's characteristics every 100 meters and thereby having the need of a large database, the model has user-defined links that have specified lengths, terrain and road surfaces. These characteristics are chosen because a wealth of information is

currently available in this form. This greatly reduces the database needed, and allows the model to be run on a stand-alone personal computer. Therefore, if no information is available from the Defense Mapping Agency concerning a particular area in the world, the user can input a best guess as to the length, terrain and road surface of a link.

4. Rates of Travel

Constant rates of travel are assumed through each link a vehicle traverses. However, each vehicle type may have a different rate that is based on the link's road surface and terrain. When vehicles travel in convoy, the slowest vehicle's rate of travel is used in determining the convoy's travel time.

5. Vehicle Breakdowns

Because links are uniform, vehicles only breakdown at nodes. Vehicles will check their status at each node and are either temporarily fixed or left to be recovered.

6. Aggregation of Cargo

Supply cargo are aggregated instead of tracking each nut and bolt in the supply system. At the basic level, classes of supply are tracked and separated by pounds. Special cargo, like tanks or infantry fighting vehicles are not separable and are individually tracked. Any cargo can be designated nonseparable and scheduled to be moved from one location to another.

7. Units

All units have constant consumption rates for any supplies the user wishes to use in the simulation. Estimated rates of consumption for the different classes of supply

for various sizes of units are published in the U.S. Army's Field Manual 101-10-1/2, *Staff Officer's Field Manual: Organizational, Technical, and Logistical Data Planning Factors (Volume 2).* [Ref. 8.]

B. MEASURES OF PERFORMANCE

ITTSS, because of its flexibility, can provide several different measures of performance to answer the following types of questions.

1. Time Required to Move Cargo

What is the amount of time required to move a large amount of nonseparable cargo from one location to another?

2. Average Amount of Cargo Moved Daily/Weekly

Given a transportation network, what is the average amount of cargo in short tons that is moved daily from motorpools to supply points?

3. Utilization Rate of Assets

How many assets are actually being used or committed daily? Are there too many assets for the mission being conducted or are more needed to successfully accomplish the mission? If cargo needs to be relocated within a certain time period, how does increasing or decreasing the number of assets effect the time needed to relocated the cargo? If the simulation is rerun and decreasing the number of assets onhand does not effect the time required to relocate, then there are more assets than needed to accomplish the mission. On the other hand, if increasing the amount of assets reduces the relocation time, then more assets are needed.

4. Availability Rate of Assets

How many assets are mission capable? This may be calculated on either a daily basis or for a specific mission.

5. Comparison of the performance of assets

How is asset performance (average pounds moved or time required to move cargo) affected by unit (assets) location, type of assets, unit size, prioritization of cargo, or selection of different routes?

C. PARAMETERS

1. Model Input Parameters

The model has many input parameters that the user can modify to allow a wide range of scenarios. A list of these parameters follows.

a. Asset parameters

(1) Performance characteristics. Rates of travel and fuel consumption are in the model's data files (Appendix C) for a variety of vehicles. Rates of travel are given for each vehicle over a particular road surface and terrain. These rates determine the length of time a vehicle takes to traverse a link. Fuel consumption rates are used to determine how much fuel is needed to support the assets. Fuel levels are maintained at each fuelpoint. If a vehicle runs out of fuel and no fuelpoint is available, the vehicle is assumed to have a reserve fuel supply to complete travel to its destination. This allows the user to identify links that may require an additional fuelpoint. If desired, the user could establish additional fuelpoints and rerun the model. The user can input different

values to see the effects that future vehicles with better performance characteristics would have on the transportation system.

(2) Reliability. Mean miles between operational mission failure (MMBF), maintenance manhours per mile and administrative and logistics downtime are listed in Appendix C for commonly used vehicles. Vehicles are given an exponentially distributed mean number of miles to breakdown based on the input MMBF. Upon breakdown, the vehicle will either continue to its destination after a brief delay for repair or be left at a node to be recovered by a wrecker. In either case, the vehicle, upon return to a motorpool will be unavailable until repaired. Once repaired, the vehicle will again be given a new mean miles to breakdown. The user can either increase or decrease the MMBF to see the impact on asset availability.

(3) Cargo load dimensions. An asset's exact cargo load dimensions of length in inches and weight in pounds are used if nonseparable cargo is to be moved. If separable, only the weight capacity of an asset is used to load cargo. Changing this input can show what impact future vehicles which have larger or smaller cargo load dimensions would have on transporting cargo within the transportation system.

b. Link parameters

(1) Distance. Distances are input in miles. Links connect 2 nodes together. A node may be a unit, fuelpoint, supply point, maintenance facility, motorpool, or any combination.

(2) Origin and destination. A specific origin and destination must be established for each link.

(3) Road surface and terrain. The numerical values for these parameters are in the database of the model. Each link must be distinguished by a certain type of road surface: concrete, bituminous (asphalt), gravel, or dirt. A particular terrain must also be chosen: flat, rolling hills, hills with curves, or mountainous.

c. Route parameters

Each route must be explicitly stated in the route data file. From origin to destination, each node that makes up the route must be listed. From this information, a link list is created that convoys traverse when travelling within ITTSS.

d. Supply parameters

(1) Supply source. A supply source must be declared for each supply point. Resupplies are requested and missions are scheduled to deliver the supplies from the supply source to the requesting source.

(2) Initial stock level. The beginning stockage level for each supply item must be set. If a value of 0 is entered, an immediate resupply will be requested with the supply point's first checking of its stock. This resupply continues until the required minimum number of days of supply for each item is onhand.

(3) Days of supply on hand. This parameter is multiplied by the unit's consumption rate to determine the minimum amount of a supply item that is required to be on hand.

(4) Day to start/end and time to check stock. These parameters determine the dates when a supply point should start and end checking its supply stocks. These values can be changed to emulate a supply point that has not yet arrived or one that will change locations sometime later in the simulation run. The time to check stock designates the time every day the supply point will check its stock.

(5) Receiving and loading points. At each supply point object a certain number of receiving and loading points can be set. Convoys will compete for these points on a first come first serve basis, or on priority if the cargo in one convoy has a higher priority than cargo in another. If all points are being used, a convoy will have to wait until a point becomes clear. The end result is that a supply point can load or unload a certain number of convoys at the same time. This process is handled by the built-in functions of the MODSIM II™ construct called ResourceObj [Ref. 7: p. 163].

(6) Material Handling Equipment (MHE). Each receiving or loading point has a specified number of MHE available. MHE unload the cargo from the assets, and determine the number of assets that can be unloaded at the same time. MODSIM II™'s ResourceObj is again utilized. If all MHE are being used, the other assets will have to wait until a MHE is available.

e. Unit parameters

(1) Unit supply. Units must have a supply point from which they draw or consume supplies.

(2) Daily consumption rate. This parameter must be established for each supply item that the unit consumes. Every 24 hours, this rate is debited from the stockage level of the unit's supply.

f. Motorpool parameters

(1) Assets. All assets are located in motorpools. The number of each kind of asset must be entered. These assets will be created before the simulation begins with the asset characteristic database. Appendix C shows an example of the database input requirement for each type of asset.

(2) Convoy Data. Distance between vehicles and convoys must be entered in feet. A maximum limit for the number of vehicles in a convoy can be set. Input for break miles and time can also be specified. During its travel, if a convoy has travelled over a certain number of miles, it will conduct a break at the next node. Inputs for standown miles and time can also be specified for convoys. Before its departure from its destination or upon its arrival on its return trip home, the convoy will standown for the specified amount of time. During Operation Desert Storm, a full day standown time was established for travel over the long routes because of the desert climate, long distance of the routes, and having only one operator available per vehicle.

The last parameters for convoys indicate whether maintenance contact teams are available and how long it takes to temporarily fix vehicles. The first input is a boolean value. If contact teams are available for travel with convoys, vehicles can be temporarily fixed by these teams and continue to their destinations. The

temporary fix time is the time required to repair each non-mission capable vehicle. Upon returning home, these vehicles will be entered into the maintenance facility and repaired. Otherwise, if contact teams are not available and a vehicle is found non-mission capable while travelling, the vehicle is left at the nearest node. The vehicle will be recovered by a wrecker and taken to either the destination node's maintenance facility or the vehicle's home maintenance facility, whichever is closest. Any cargo on the vehicle will be off-loaded either at the destination node's receiving point or at the vehicle's home motorpool and rescheduled for another mission.

g. Maintenance parameters

(1) Maintenance manhours per mile. This rate is multiplied by the number of miles a vehicle has travelled since its last breakdown. The resulting time is the length of time it takes to repair the vehicle in the maintenance facility.

(2) Administrative and logistics downtime. This is a flat rate that determines how long an asset remains non-mission capable excluding the maintenance manhours. This includes recovery time, waiting for parts, tools and anything else not included within the maintenance manhours.

h. Fuelpoint parameters

(1) Starting level. Although a starting level can be given to each fuelpoint, currently there is no effect in the model if this amount is exceeded. However, ITTSS does keep track of total fuel consumption for each fuelpoint so that the user can

estimate how much fuel is required at each fuel point. Future embellishments of the model could include the effects of running out of fuel.

(2) Number of pumps. This parameter determines how many assets can be refueled at the same time. Simulation time elapses during refueling and while assets wait for pumps to become free.

(3) Refueling time. The time required for the refueling of an asset is input here.

D. REPORTS

Four reports are currently available after each simulation run of ITTSS. Additional reports could be created for specific scenarios and easily implemented within the model's code. Examples of these reports are listed in Appendix B.

1. Dispatch Report

All convoy activity is listed within this report. A convoy is designated by its origin and time of departure. The time when a convoy leaves its origin, arrives at its destination and ultimately returns home is recorded in the Dispatch Report. The number of assets within the convoy, and the number that broke down during the mission and were temporarily repaired and remained with the convoy are also listed with each reference of convoy movement.

2. Supply Report

Every 24 hours, a supply report is generated showing the current stockage levels of all supply items within the inventory of each supply point. These include

classes of supply and nonseparable items such as tanks and infantry fighting vehicles that may have been relocated to the supply point's location.

3. Vehicle Status

A vehicle status report is generated every 24 hours showing the number of assets available and deadlined at each motorpool. Availability and utilization rates are calculated for the 24 hour period of the report. A report for each different type of asset located within the motorpool is included in the daily vehicle status report.

4. Maintenance Activities

All pertinent maintenance activities are listed within this report. The actual time an asset breaks down is reported to the asset's home maintenance facility and recorded in this report. The times that a maintenance facility receives a deadlined asset and completes its repair is also listed in this report.

E. SIMULATION MODEL

1. MODSIM II™

ITTSS was written entirely on a personal computer (486DX, 25mhz), using the OS/2 operating system. The model is a stand alone simulation that requires only the model code, MODSIM II™, Microsoft C, OS/2 1.2 and database files. ITTSS consists of a total of 54 definition and implementation modules and one main module which can be found in Appendix D. At present the only interactive feature available is for the choice of running a complete simulation or just scheduled missions. Changes to parameters must be made in the database files. This chapter contains a description of

events that drive ITTSS and actions that objects in the model perform. Because ITTSS is an object-oriented model, the actions that are described are the actual methods that are defined for the objects. When a motorpool schedules missions, a convoy travels, or an asset refuels, they all refer to objects that actually have these methods written for them. Much of the computer code for this model, listed in Appendix D, is much more readable than traditional computer languages like FORTRAN or PASCAL. Current transportation doctrine was followed as much as possible in the development of the actions that objects in ITTSS perform while allowing flexibility for a variety of future scenarios.

2. Event Driven

Two types of events drive the simulation model. The first event type is driven by the daily operations of units consuming supplies and supply points checking their stock for shortages. Missions are scheduled based on these shortages and assets travel to fill mission requirements. The second event type occurs when the user schedules missions to move cargo from one location to another. Either event can be run separately or together. The advantages in having this option is that the user can compare the performance of vehicles moving cargo from one location to another with or without the interference of other vehicles within the transportation system.

a. Daily Operations

(1) Unit consumption. Each unit has a supply object (point) from which supplies are consumed. A unit has a daily consumption rate for each supply item. At a specified time, the unit consumes the amount and thereby decreases the inventory of its supply. The default time is the end of each day. Inputs for the dates when the units are to begin and end the daily consumption of their supplies enable the user to simulate units entering or leaving the theater of operations at different times.

(2) Checking stock. Each supply object checks its inventory at a specified time each day. If stock levels fall below a particular level, the object sends a request list to its supply source for a resupply. The supply source, in turn, checks its own inventory. If items on the request list cannot be filled, the supply source will ask for a resupply from its higher supply source. Any items on the request list that are onhand are immediately scheduled by the supply source's motorpool to be delivered to the requesting supply object. Any items that are not onhand will be delivered when the supply source receives its resupply from its higher supply source.

b. Scheduled Missions

In addition to supply missions that are scheduled within the simulation to fill a resupply request, missions can be scheduled by the user to occur at any time during the simulation run. An example of a user-scheduled mission is a one-time lift of a heavy maneuver force or just the relocation of a few tanks. User-scheduled missions

can occur by themselves without interference of other supply/transportation actions or in conjunction with them.

3. Conducting Missions

Whenever a mission is scheduled, either through a resupply request or for a specified movement of cargo, certain actions occur that drive the simulation. Cargo must be loaded upon the appropriate assets, assets must be placed in convoys, and convoys must travel to a destination, perform their business and return home to complete the mission. The following sections describe in detail the actions within these operations and the purpose for the many parameters and inputs available in the model.

a. Loading cargo on assets

Once a mission is scheduled, a cargo list is sent to the appropriate motorpool. The motorpool takes each item off the list and matches it to an available asset within its asset list through a standard sorting algorithm that insures the best fit using the available assets. If the item is separable, it can be broken down into smaller pieces and just the weight is used to match cargo to the asset. If the cargo is nonseparable, both length and weight are used to find the best fitting asset to carry the cargo. The smallest asset that can carry the cargo is always chosen. If the cargo is too large for any available asset, it is separated if possible. The user must insure that there is an asset available in the motorpool that can transport any nonseparable item in the transportation system. Any cargo that is not matched to an asset is placed in a waiting-to-be-loaded queue. This queue is checked by all returning assets. The motorpool then

conducts the loading of cargo onto assets based on loading point availability and MHE availability. Loading may be delayed if these points are busy with the loading of other convoys. Once loading has begun on a convoy, the number of MHE available at the supply point will determine how many assets can be loaded at the same time.

b. Convoy Organization

All loaded assets are organized by convoy. Convoys have a user-defined maximum number, distance between assets and distance between other convoys. All assets are refueled before traveling and a determination is made whether or not all assets can make it to their destination and back without refueling. If they can, then the convoy will not stop for any fuel throughout the mission. The user can define whether or not a maintenance contact team/wrecker is available to travel with the convoy. If a contact team is available, vehicles that breakdown during the mission are temporarily repaired until they return home. Upon return they are entered into their maintenance facility for repairs. If wreckers are not available, vehicles that breakdown during the mission are left at a node and a wrecker from either the home or destination point, whichever is closer, will recover the asset and tow it to the maintenance facility. Cargo will either be unloaded at the destination's supply point or at the origin's motorpool where another mission will be scheduled. When a convoy is ready for travel, it is released after the appropriate distance between it and the previous convoy has been cleared.

c. Traveling

Convoys travel along a route that contains a list of links. The speed of the slowest asset in the convoy is used to determine the time it takes to travel across the link. Vehicles can reach their miles between failure during this time, but will not become non-mission capable until they reach the next node. Miles driven and fuel consumed are debited after the convoy enters a node.

(1) Entering a node. At the end of each link are nodes that may contain supply, motorpool, maintenance or fuelpoint objects. Upon arrival at the node, the convoy will determine whether or not the node is its destination for a resupply mission, recovery of vehicles, or for the return of a repaired vehicle by another maintenance facility.

If the vehicles carry cargo and it is at its destination node, the convoy will enter the node's supply point. Unloading of the cargo is similar to loading except convoys that carry higher priority cargo are be offloaded first by the available receiving points and MHE. If the vehicles in the convoy are wreckers, any broken-down vehicles that are at the node will be recovered and carried back to a maintenance facility. If the convoy contains repaired vehicles, these vehicles will be returned to their motorpools.

Regardless of whether or not the node is a destination node, all vehicles update their odometers and fuel gauges, check for breakdowns and refuel if necessary. Delays occur if the convoy refuels, temporarily fixes a vehicle, or conducts business within a destination node. Before departure from the node, the convoy checks

if its miles travelled is greater than its maximum allowable miles before a break is required. If the miles travelled exceed this limit, the convoy breaks for the required time. If the convoy is at a destination node and is about to return to its home motorpool, a standown time will delay the convoy the appropriate length of time.

After these checks, the convoy must receive clearance before passing through the node. Only one clearance is available at a time. If more than one convoy is at the node, the first one that arrived will be allowed to proceed. Once the appropriate distance between convoys has been reached, the next convoy will be allowed to continue its travel. Unless a convoy has a user-specified higher priority, convoys are released to travel by the order they arrive when they request clearance.

(2) Returning Home. Convoys return to their home motorpool along a return route of links. The same actions of updating gauges and checking for refueling and breakdowns are performed at each node they cross. Upon return to their home node, preventative maintenance is performed on each vehicle in the form of checking for breakdowns and refuelling. Any vehicles that are non-mission capable are entered into the maintenance facility for repair. A standown time may be necessary if the convoy has travelled farther than the allowable mileage. Once these actions have been conducted, cargo that is waiting to be loaded is checked. If the vehicles can carry the cargo, another mission is conducted. If there is no match or no more cargo to haul, the vehicles are finally returned to their motorpool until another mission is scheduled.

IV. MODEL DEMONSTRATION EXAMPLE

A. INTRODUCTION

This chapter presents a demonstration of ITTSS's capability to estimate the time required to move cargo from one location to another. This demonstration compares the new doctrine of relocating a heavy maneuver force in a single lift using the new HETs with the old doctrine using the actual vehicles available in Operation Desert Storm. During the war, HETs were so critical that they were "intensively managed at the General officer level on a daily basis, and during critical periods on an hourly basis" [Ref. 3: p. 2]. What is of interest is the comparison of the expected travel time for the two doctrines.

B. SCENARIO

The single lift of a HMF in the Operation Desert Storm scenario is used to show how the measure of performance, time required to move the HMF from one location to another, can be estimated from ITTSS. From ITTSS's output, data can be obtained to perform a paired-sample test to show how much better the new doctrine is compared to the old doctrine. Table 2 shows a typical brigade size heavy maneuver force (HMF) that could be expected to be moved by the new transportation corps HET doctrine.

ITTSS runs will be made using the same northern route that was actually used to move the tracked vehicles during Operation Desert Storm. The origin will be Dammam

and the destination will be King Khalid Military City (KKMC). See Figure 3. Only the new 70 ton HETs will be used for runs involving the new doctrine, since they are the only HETs considered in the new single-lift concept [Ref. 4: p. 41]. In Operation Desert Storm, the Army did not have any of the new 70 ton HETs to move the larger tracked vehicles like the M-1. HETs had to be contracted from other countries to fill this void [Ref. 3: p. 2]. These contracted HETs and the Army's 60 ton HETs are used in runs involving the old doctrine.

TABLE 2. TYPICAL HEAVY MANEUVER FORCE.

TRACKED VEHICLE TYPE	NUMBER TO BE MOVED
M1 Abrams, Main Battle Tank	116
M2 Bradley, Infantry Fighting Vehicle	64
M3 Bradley, Cavalry Fighting Vehicle	18
M106 107mm Self Propelled Motor *	6
M109 155mm Self Propelled Howitzer	24
M113 Armored Personnel Carrier (APC)	118
M548 Cargo Carrier *	30
M578 Armored Recovery Vehicle	3
M577 Armored Command Vehicle *	29
M901 Improved TOW Vehicle *	12
ACE Armored Combat Earthmover	27
AVLB Mechanized Bridge	12
CEV Combat Engineer Vehicle	6
FIST-V Fire Support Team Vehicle *	16
* very similar to the M113, APC	**505 Total Tracks**

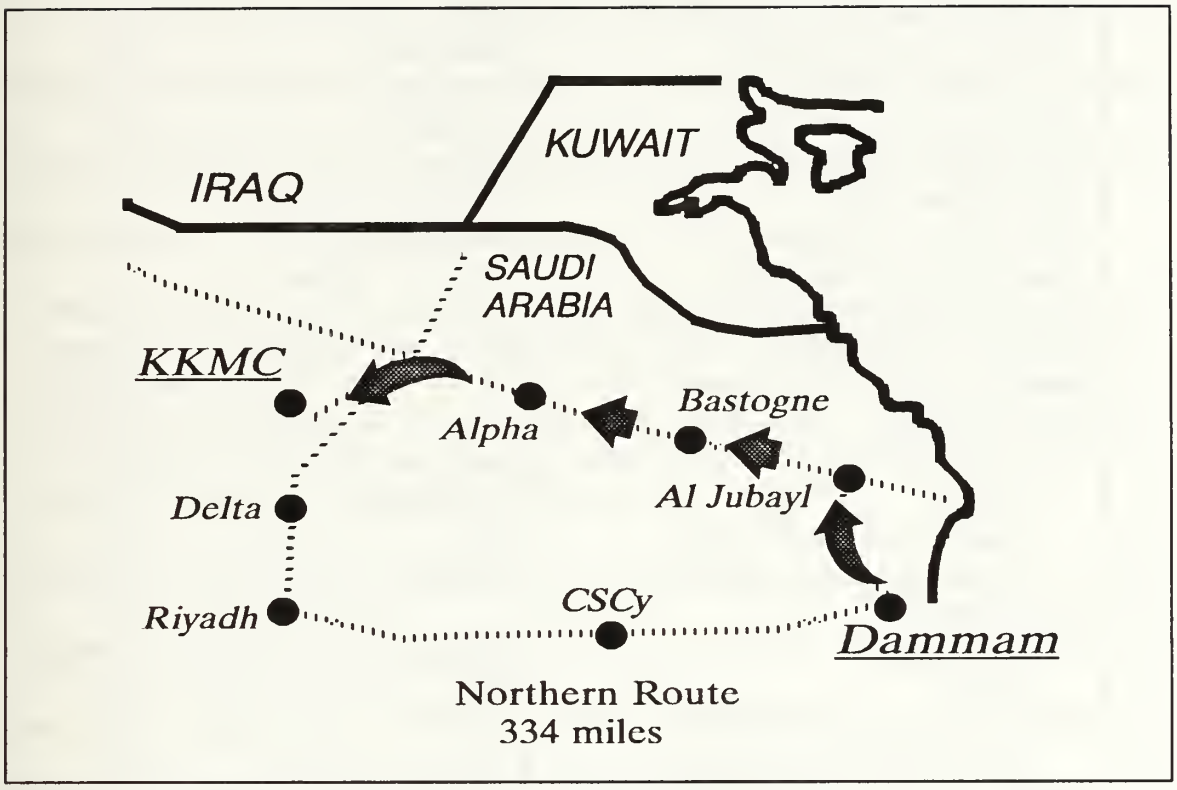

Figure 3. Routes During Operation Desert Storm

The significant model parameters for both runs are listed in Table 3. Appendix C contains the actual data files that were used for both runs. The result of these runs will provide point estimates for the travel time from Dammam to KKMC for each doctrine.

TABLE 3. MODEL PARAMETERS.

PARAMETER	VALUE
Maximum Number in Convoy	25
Distance Between Convoys	3000 feet
Distance Between Vehicles	350 feet
Maintenance Contact Teams	Available
Temporary Fix Time	15 min
Break Distance/Time	90 miles / 30 min
Standown Distance/Time	200 miles / 1 hr
Supply Load/Unload Time	15 min / 15 min
Supply Loading Points	2
Supply Receiving Points	2
Material Handling Equipment	5
New Doctrine:	
HET 70 Ton	385 total
MMBF	2500 miles
Maint Manhours Per Mile	.005
Admin and Log Downtime	34 hrs
Old Doctrine:	
HET 60 Ton	160 total
MMBF	1600
Maint Manhours Per Mile	.009
Admin and Log Downtime	34 hrs
Contracted HET, 70 Ton	120 total
MMBF	1600
Maint Manhours Per Mile	.009
Admin and Log Downtime	34 hrs

No data is available for the contracted HETs. An assumption is made that they have the same performance capabilities as the current 60 Ton HETs of the Army, except they can carry heavy loads that the 60 Ton HETs cannot.

C. SIMULATION RUNS

100 runs each were made using the new and old doctrines. The variance-reduction technique of common random numbers was used in setting up the simulation runs. Variance reduction techniques can increase the model's efficiency by reducing the variance of the estimated travel times without disturbing its expectation, and also produce smaller confidence intervals for the difference between the two doctrines. Common random numbers are used when comparing two alternative system configurations.

> We want to compare the alternative configurations [the 2 doctrines] "under similar experimental conditions" so that we can be more confident that any observed differences in performance are due to differences in the system configurations rather than to fluctuations of the "experimental conditions". In simulation, these "experimental conditions" are the generated random variates that are used to drive the models through simulated time. [Ref. 10: pp. 612-613]

All initial conditions were identical. 100 random number seeds were used to generate a separate stream of 500 random numbers that were used for each individual simulation run. These streams were created by MODSIM II™'s random number generator which is a multiplicative congruential pseudo-random number generator [Ref. 9: p. 160]. The generator allowed the exact random number stream used in each simulation run of the new doctrine to be used in the corresponding run of the old doctrine. Every vehicle received the same MMBF input in each corresponding run. This parameter was used to produce each vehicle's miles to failure from an exponential distribution. This enabled as much matching up of random numbers across the different doctrines on a particular replication as possible. The first 280 vehicles in each set of simulation runs received the same random numbers to determine their miles before failure.

A single simulation run took approximately 4 minutes of computer time. The only change in the input to the simulation during each run was the random number stream. Because initial starting conditions were identical for each run, the same number of vehicles were loaded. The departure time of the first convoy was also constant in every one of the simulation runs. The expected travel time was determined by when the first convoy of HETs left Dammam and the last one arrived at KKMC. Because of delays in having to temporarily fix HETs that break down, and in waiting for clearance to travel, a convoy that left first did not necessarily arrive first.

D. SIMULATION RESULTS

Because of the terminating nature of the simulation runs (one mission to move cargo from Dammam to KKMC), no steady state analysis was necessary. Each run was independent, terminated by the arrival of the last convoy at KKMC and begun again with identical initial conditions.

1. Data Analysis

Three sets of data were obtained from the 200 total simulation runs (Appendix A). A travel time from Dammam to KKMC was obtained for each run using the new and the old doctrines. The difference between the times was calculated for each run. These will be used in paired-sample analysis. X-Y plots in Figure 4 shows the data taken from the model's output. The top plot displays the old doctrine travel times. Below it are the paired-differences between the doctrines. The bottom plot shows the travel times using the new doctrine.

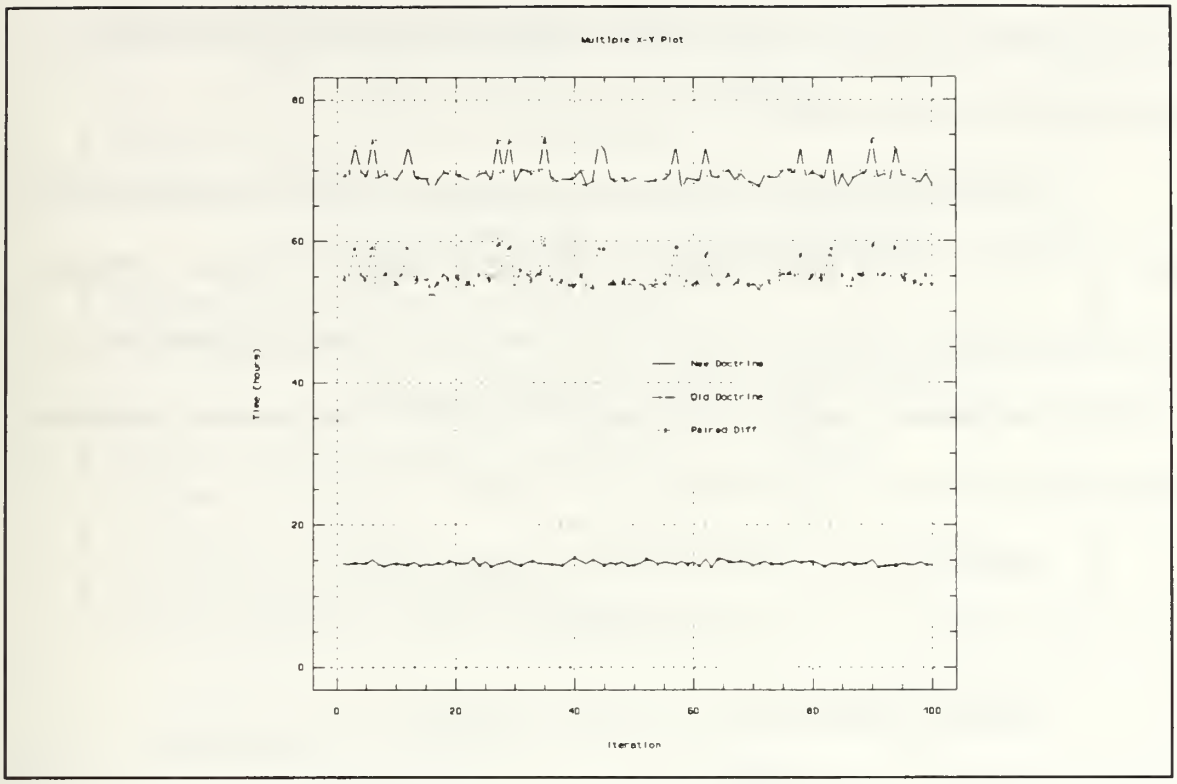

Figure 4. Data from the 200 Runs

2. Paired-Samples Statistics

The following statistics were used in finding point estimates, where X_i is the performance of old doctrine in the ith simulation run [Ref. 10: pp. 532-533].

Sample Mean:

$$\bar{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n},$$

Variance:

$$S^2(n) = \frac{\sum_{i=1}^{n} [X_i - \bar{X}(n)]^2}{n-1}$$

The sample mean and variance for Y_i, the performance of the new doctrine in the ith simulation run are found similarly.

The 100 independent replications of the simulation resulted in each of the data points, X_i and Y_i, being independent and identically distributed random variables. Therefore, the sample mean is an unbiased point estimator for the travel time. [Ref. 10: p. 532] The formulas used in determining the estimate of the expected difference between the doctrines are listed below [Ref. 12: p. 49].

Estimate of the expected difference in performance:

$$D_i = X_i - Y_i \ ,$$

Confidence interval for the expected difference:

$$\bar{D} \pm k\sqrt{\frac{s_D^2(n)}{n}} \ , \textit{ where } k \textit{ is the } t\textit{-statistic}$$

Variance of the expected difference:

$$Var[D_i] = Var[X_i] + Var[Y_i] - 2Cov[X_i, Y_i]$$

The t-distribution was used to estimate a 95% confidence interval with α = .05. With 95% confidence, the estimates for travel time and availability will be within this interval. A t-value of 1.96 was used for v (v=n-1, the degrees of freedom) equalling 99 [Ref. 11: p. 20]. Table 4 summarizes the results for point estimates, variances, and confidence intervals for both doctrines.

TABLE 4. SIMULATION RUN STATISTICS.

NEW DOCTRINE (70 Ton HETs)
Estimate of Travel Time = 14.5916 hours Variance = .073918
OLD DOCTRINE (Old and Contracted HETs)
Estimate of Travel Time = 69.6946 hours Variance = 3.09445
PAIRED SAMPLE STATISTICS
Estimate of Difference Between Doctrines, D = 55.103 Variance = 3.04108 Covariance = .06365 Correlation Coefficient = .1331 Confidence Interval = [55.685,58.520]

3. Simulation Run Results

The travel time estimators in Table 4 seem reasonable for heavy equipment transporters travelling routes of over 300 miles. The estimate of difference between the doctrines is just over 55 hours. Applying the 95% confidence interval, we can be 95% confident that the true difference is between 51.685 and 58.520. The use of the paired-sample approach and common random numbers did reduce the variance. Using common random numbers can positively correlate estimators and increase the covariance between them which reduces the variance of the estimate of the difference between them.

V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSIONS

There are no simulation models currently available that treat transportation assets as individual objects and examine the variability in the reliability of assets. ITTSS fills this modelling void by providing the capabilities to focus on the performance and reliability characteristics of transportation assets. Many large combat models aggregate transportation assets and therefore make it impossible to analyze the individual performance of these assets. Still other models, like the Distribution System Analyzer (DSA), provide a very detailed modelling of the supply and transportation system, but intensive knowledge of the supply system is essential in setting up each scenario. DSA provides a good analysis of the movement of supply items within the supply system, but does not allow us to focus on transportation assets like ITTSS. ITTSS gives the user an option to model an entire transportation system and/or to schedule movements of cargo. Being an object-oriented simulation model, ITTSS allows individual monitoring of assets throughout the simulation run and flexibility to easily add other types of assets.

The results produced by ITTSS from the model demonstration problem are very reasonable. The travel times of 14.5 and 69.7 hours are reasonable time estimates for HETs traveling in convoy over routes greater than 300 miles, given the doctrines that were followed.

ITTSS allows the user to focus on transportation assets' performance and reliability characteristics and can easily be modified or augmented to fit a wide range of future scenarios or doctrinal changes without significant computer code changes.

B. RECOMMENDED FUTURE ENHANCEMENTS

The following enhancements for ITTSS are recommended for future development and are currently being pursued by the author.

1. User-Friendly Menu

A user-friendly, menu-driven input capability will simplify data entry. Currently all data must be entered through data files. The graphic capabilities of SIMGRAPHICS® can be integrated with ITTSS MODSIM II™ code to provide a more foolproof and error free method of entering and changing parameters.

2. Other Types of Assets

Other types of assets could be defined within the transportation system. Aircraft, especially helicopters, are an integral part of an intra-theater transportation system. To take advantage of existing rail systems and inland waterways, train and vessel objects could also be introduced.

3. Separation of Tractors and Trailers

While the assumption of aggregating tractors and trailers was made for this model, a future enhancement to the model would be to separate them. This would enable trailer transfer point operations to be accurately modelled. Army doctrine might advocate the shuttling of trailers from one location to another by tractors that are

dedicated to a segment of the route. The trailers would then be shuttled by another group of tractors dedicated to the next segment in the route.

4. Containers

Containers could also be introduced to take advantage of ITTSS's object-oriented view. During Desert Storm, thousands of containers were within the theater of operations and the managing of these assets was overwhelming. Future conflicts will most certainly involve a great number containers to move unit equipment and classes of supply.

5. Combat Losses

Random ambushes or air attacks that impact on the performance of assets and the movement of cargo and that partially or fully disable a link or node could be introduced into ITTSS.

LIST OF REFERENCES

1. Churchill, W. S., *The River War*, Thomas Nelson and Sons, 1899.

2. Van Creveld, M., *Supplying War*, Cambridge University Press, 1977.

3. 22nd Support Command(TAA), *Theater Linehaul Transportation Operations During Desert Shield and Desert Storm*, by Major Paul L. Willis, 1991.

4. Fortner, J., Doux, J., Peterson, M., "Bring on the HETs! Operational and Tactical Relocation of Heavy Maneuver Forces," *Military Review*, pp 36-45, January 1992.

5. Department of the Army, FM 55-15, *Transportation Reference Data*, 1986.

6. Foss, C., Gander, T., *Jane's Military Vehicles and Logistics*, Jane's Information Group, 1992.

7. U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, Subject: Reliability Data, 22 June 1991.

8. Department of the Army, FM 101-10-1/2, *Staff Officer's Field Manual: Organizational, Technical, and Logistical Data Planning Factors (Volume 2)*, 1986.

9. CACI Products Company, *MODSIM II™Reference Manual*, CACI Products Company, 1991.

10. Law, A. M., and Kelton, W. D., *Simulation Modeling and Analysis*, McGraw-Hill, Inc., 1991.

11. Neave, H. R., *Elementary Statistics Tables*, Biddles Ltd, 1981.

12. Bratley, P., Fox, B. L., and Schrage, L. E., *A Guide to Simulation*, Springer-Verlag, 1987.

13. Hillier, F. S., and Lieberman, G. J., *Introduction to Operations Research*, Holden-Day, Inc., 1980.

14. Koopmans, L. H., *Introduction to Contemporary Statistical Methods*, Duxbury Press, 1987.

15. U.S. Army Corps of Engineers, Waterways Experiment Station, Vicksburg, MS, Subject: Vehicle Travel Rates, 7 August 1991.

16. CACI Products Company, *MODSIM II™The Language for Object-Oriented Programming-User's Manual*, CACI Products Company, 1991.

APPENDIX A - DATA FROM EXAMPLE PROBLEM

Table 5. Data from Simulation Runs.

RUN i	TRAVEL TIME Old Doctrine X_i	TRAVEL TIME New Doctrine Y_i	DIFFERENCE X_i-Y_i
1	69.17737	14.53439	54.64299
2	69.50056	14.42781	55.07275
3	73.55607	14.63	58.92607
4	69.92296	14.53439	55.38858
5	69.22592	14.59315	54.63276
6	74.36455	15.1061	59.25845
7	68.99762	14.45171	54.5459
8	69.57631	14.21267	55.36364
9	69.18399	14.46267	54.72132
10	68.69243	14.53439	54.15805
11	69.96341	14.43877	55.52464
12	73.58925	14.38	59.20925
13	69.19834	14.68877	54.50957
14	68.9553	14.26048	54.69482
15	68.95422	14.42781	54.52641
16	67.17443	14.38	52.79443
17	68.70383	14.55829	54.14554
18	69.7546	14.41487	55.33974
19	69.5324	14.88	54.65239
20	69.49688	14.67781	54.81907
21	69.22592	14.53439	54.69153

RUN i	TRAVEL TIME Old Doctrine X_i	TRAVEL TIME New Doctrine Y_i	DIFFERENCE X_i-Y_i
22	68.74909	14.59315	54.15593
23	68.73362	15.28439	53.44923
24	69.57595	14.30829	55.26766
25	69.71744	14.80829	54.90915
26	68.73511	14.14295	54.59216
27	74.31042	14.45171	59.85871
28	69.72626	14.62999	55.09627
29	74.07509	14.92781	59.14728
30	68.42149	14.45171	53.96978
31	70.22739	14.30829	55.9191
32	70.03109	14.70171	55.32937
33	69.44243	14.89096	54.55147
34	70.18581	14.58219	55.60362
35	74.78137	14.58219	60.19918
36	69.04616	14.45171	54.59444
37	68.45201	14.40391	54.04811
38	68.76194	14.30829	54.45365
39	68.7318	14.92781	53.80399
40	69.00276	15.33219	53.67057
41	70.00754	14.93877	55.06878
42	68.04894	14.53439	53.51456
43	68.4259	15.05829	53.36761
44	73.78509	14.70171	59.08338
45	73.01715	14.30829	58.70886
46	68.78513	14.63	54.15512

RUN i	TRAVEL TIME Old Doctrine X_i	TRAVEL TIME New Doctrine Y_i	DIFFERENCE X_i-Y_i
47	68.43399	14.45171	53.98228
48	69.00056	14.70171	54.29884
49	68.49541	14.24952	54.24589
50	68.92002	14.3561	54.56392
51	68.79616	14.58219	54.21396
52	68.44832	15.14096	53.30735
53	68.47663	15.01048	53.46615
54	68.75203	14.42781	54.32422
55	68.76267	14.73658	54.0261
56	69.46709	14.65391	54.81319
57	73.58523	14.41487	59.17037
58	67.49467	14.83219	52.66248
59	68.91928	14.43877	54.48051
60	68.72739	14.73658	53.99081
61	68.54616	14.23658	54.30958
62	73.31427	15.15391	58.16036
63	69.20422	14.03439	55.16984
64	69.00203	15.17781	53.82422
65	69.75975	15.15391	54.60585
66	69.98364	14.76048	55.22316
67	68.68179	14.70171	53.98007
68	69.48141	14.8322	54.64921
69	68.45199	14.71267	53.73932
70	68.27299	14.30829	53.9647
71	67.75679	14.53439	53.2224

RUN i	TRAVEL TIME Old Doctrine X_i	TRAVEL TIME New Doctrine Y_i	DIFFERENCE X_i-Y_i
72	69.29248	14.83219	54.46029
73	69.00497	14.49952	54.50545
74	68.96523	14.48658	54.47865
75	69.94539	14.42781	55.51758
76	70.17848	14.76048	55.418
77	69.67664	14.92781	54.74883
78	72.81427	14.67781	58.13646
79	69.19243	14.78439	54.40805
80	69.71709	14.78439	54.93271
81	69.23546	14.51048	54.72498
82	68.94464	14.11706	54.82758
83	73.28515	14.58219	58.70296
84	67.22075	14.55829	52.66246
85	69.45348	14.40391	55.04958
86	67.77005	14.68877	53.08128
87	69.0204	14.45171	54.56868
88	69.47592	14.43877	55.03715
89	69.72702	14.58219	55.14483
90	74.58108	15.1061	59.47498
91	69.0605	14.08219	54.97831
92	69.46376	14.23658	55.22718
93	68.7296	14.3561	54.3735
94	73.29292	14.28439	59.00853
95	69.25423	14.53439	54.71985
96	69.28069	14.41487	54.86583

RUN i	TRAVEL TIME Old Doctrine X_i	TRAVEL TIME New Doctrine Y_i	DIFFERENCE X_i-Y_i
97	68.69611	14.40391	54.2922
98	68.26343	14.68877	53.57466
99	69.49247	14.40391	55.08856
100	68.16928	14.40391	53.76538

APPENDIX B - MODEL OUTPUT FROM EXAMPLE PROBLEM

I. DISPATCH REPORT

The dispatch report contains information on the departure, arrival and return of all convoys. A convoy is identified by its motorpool and time of departure. The time and place of the action are listed next. The last item is how many assets were in the convoy (convoys can leave non-mission capable assets at a node if no maintenance contact teams are available). Following are excerpts of what an actual simulation run produced.

```
**************************************
**********CONVOY ACTIVITIES************
**************************************

Convoy DammamMotor60.073864 is leaving DammamMotor
  TIME OF DEPARTURE is 60.073864.  Number in convoy is 25

Convoy DammamMotor60.110795 is leaving DammamMotor
  TIME OF DEPARTURE is 60.110795.  Number in convoy is 25

CONVOY DammamMotor60.073864  arrived at KKMC
  TIME OF ARRIVAL is 73.386943
  Number in convoy is 25.

CONVOY DammamMotor60.110795  arrived at KKMC
  TIME OF ARRIVAL is 73.910847
  Number in convoy is 25.

convoy DammamMotor60.110795  returned to KKMC
  time of RETURN is 106.657896
  Number in convoy is 25.
```

II. MOTORPOOL REPORTS

Every 24 hours a summary is generated for each motorpool in ITTSS. For each vehicle type that a motorpool contains, the total assets, number committed, available and deadlined are shown. The utilization and availability rates are also listed.

```
*******************************
*******MOTORPOOL REPORTS****

************************************************************
*******Status for DammamMotor MOTORPOOL.  Time is 120.000000
----------Vehicle Type is HETTS-----------------------
Total Assets are 385
Total Committed is 272   Util  Rate % is        0.706494
Total Available is 272   Avail Rate % is        0.706494
Total Deadlined is 113
************************************************
----------Vehicle Type is  WRECKERS----------
Total Assets are 4
Total Committed is 0    Util  Rate % is        0.000000
Total Available is 4    Avail Rate % is        1.000000
Total Deadlined is 0
************************************************
```

III. MAINTENANCE ACTIVITIES

Whenever a maintenance action occurs, it is outputted to the activities report. An asset's breakdown is reported with a reference to the time and location of the incident. Whenever an asset is received to be repaired or returned repaired by a maintenance facility, the action is listed in the report. An excerpt from one of the simulation runs is found below.

```
*********************************
*******MAINTENANCE ACTIVITIES****

Asset NMC at AlJubayl.  Bumper Number is Hett_ Dammam_326
   Time of breakdown is 62.543533
```

Asset NMC at AlJubayl. Bumper Number is Hett_ Dammam_272
Time of breakdown is 62.591341

Maint DammamMaint received Hett_ Dammam_291 at 109.891540

Maint DammamMaint received Hett_ Dammam_293 at 109.891540

MAINT DammamMaint RELEASED Hett_ Dammam_6 at 143.997896

MAINT DammamMaint RELEASED Hett_ Dammam_10 at 143.997896

IV. SUPPLY REPORTS

These reports are generated every 24 hours for each supply point in ITTSS. For every separable supply item, the current stock level, days of supply onhand required, maximum upper level, daily consumption rate and the current amount on order is listed. For nonseparable items, like tracked vehicles, only the amount onhand is output.

****
********SUPPLY REPORTS**********

****
*******Status for DammamSup SUPPLY POINT. Time is 96.000000
----------Record for CL5----------
stock level is 1000000.000000
days of supply is 5
upper level is 10000000.000000
daily consumption is 1000.000000
on order is 0.000000
****
****
*******Status for KKMCsup SUPPLY POINT. Time is 96.000000
----------Record for CL5----------
stock level is 1000000.000000
days of supply is 5
upper level is 10000000.000000
daily consumption is 1000.000000
on order is 0.000000
****

----------Record for M1------
number on hand is 93
****
----------Record for M2------
number on hand is 64
****
----------Record for M113----
number on hand is 118
****
----------Record for M109----
number on hand is 8
****
----------Record for ACE-----
number on hand is 27
****
----------Record for M548----
number on hand is 30
****
----------Record for M106----
number on hand is 6
****
----------Record for FIST-V--
number on hand is 16
****
----------Record for M577----
number on hand is 29
****
----------Record for M578----
number on hand is 3
****
----------Record for M3------
number on hand is 18
****
----------Record for M901----
number on hand is 12
****
----------Record for M88-----
number on hand is 24
****
----------Record for AVLB----
number on hand is 12
****
----------Record for CEV-----

```
number on hand is  6
**********************************
************************************************************
```

APPENDIX C - INPUT DATA FILES

Data for ITTSS must be input by means of data files that are listed in this appendix. The method of reading the data and inputting it for use in the model was provided by Professor Michael P. Bailey of the Naval Postgraduate School. This method requires the data to be listed in a particular order. If too few or too many parameters are listed or spaces between parameters are omitted, run-time errors could occur. Aside from the MASTER.DAT file, each data file begins with an integer that represents how many topstrings, the names of records in the array, will be read into that data file's input array. The topstrings are the names that are located before the "->". They represent the name of individual records that contain ownedstrings. Ownedstrings are the characters after the "->". The ownedstrings represent the data in each record. Data in each record is accessed by using the topstring to find the correct record and using the position of the ownedstring to find the data's position in the data array. After each data file is a description within braces of what each position of the topstring and its ownedstrings represent.

The future version of ITTSS will be menu driven through the use of SIMGRAPHICS. The user will have limited access to the data files and therefore, fewer errors will occur when parameters need to be updated or changed.

I. MASTER.DAT

```
NODE.DAT
MOTORPL.DAT
SUPPLY.DAT
SSOURCE.DAT
FUELPT.DAT
UNITS.DAT
VEHFLD.DAT
TRAVELR.DAT
VEHOWN.DAT
LINKS.DAT
ROUTES.DAT
MISSION.DAT
SEEDS.DAT
```

II. NODE.DAT

```
9
Economy -> 300  18  90  .5  1  5
        M none      MA none      S TheEconomy
        F none                U  EUnits  EOF \\
Dammam -> 300  18  90  .5  1  5
        M DammamMotor MA DammamMaint S DammamSup
        F DammamFuel  U  DammamUnits  EOF \\
AlJubayl -> 300  18  90  .5  1  5
        M none      MA none      S none
        F none      U  none  EOF \\
Bastogne -> 300  18  90  .5  1  5
        M none      MA none      S none
        F none      U  none  EOF \\
ALPHA ->  300  18  90  .5  1  5
        M none      MA none      S none
        F none      U  none  EOF \\
KKMC ->  300  18  90  .5  1  5
        M none      MA none      S KKMCsup
        F KKMCfuel    U  KKMCunits  EOF \\
Riyadh ->  300  18  90  .5  1  5
        M none      MA none      S none
```

```
        F RiyadhFuel  U  none  EOF \\
DELTA ->    300   18   90  .5   1   5
        M none        MA none        S none
        F none        U  none  EOF \\
CSCy ->     300   18   90  .5   1   5
        M none        MA none        S none
        F CSCyFuel    U  none  EOF \\

{node name -> maxMiles before standDown, standDownTime,
          milesBeforeBreak, breakTime, dayToStart, dayToEnd
          motorpool, maintenance facility, supplypoint,
          fuelpoint, unit}
```

III. MOTORPOOL.DAT

```
1
Dammam ->   25     350    1000    TRUE    .5  EOF \\

{node name -> maxNumbInConvoy, distBetweenVeh (feet), distBetweenConvoys (feet),
      wreckersConvoy (maint contact team avail), repairTime}
```

IV. SUPPLY.DAT

```
3
Economy  -> 7.0  2   2   2   .25   .25
        CL5  CARGO  0   1000000   5  10000000  EOF  \\

Dammam   -> 7.0  2   2   2   .25   .25
        CL5  CARGO  0   1000000   5  10000000  EOF  \\
KKMC     -> 7.0  2   2   2   .25   .25
        CL5  CARGO  0   1000000   5  10000000  EOF  \\

{node name  -> TimeToCheckStock, # of ReceivingPoints, # of LoadingPoints,
                  # of MHE, loadTime, unloadTime
             Item, MOR, Priority, DailyConsRate, StkLvl, DOS, UpLvl}
```

V. SSOURCE.DAT

```
3
TheEconomy -> TheEconomy  \\
DammamSup  -> TheEconomy  \\
KKMCsup    -> DammamSup   \\
```

{supply point name -> supply source}

VI. FUELPT.DAT

```
4
Dammam -> 0.0
          5000.0  .25  4  \\
KKMC   -> 0.0
          5000.0  .25  3  \\
Riyadh -> 0.0
          5000.0  .25  3  \\
CSCy   -> 0.0
          5000.0  .25  3  \\
```

{node name -> Starting Fuel Level
Max Fuel Level, Refueling Time, Number of Fuel Pumps}

VII. UNITS.DAT

```
3
Economy ->  1  5  CL5  1000  EOF  \\
Dammam  ->  1  5  CL5  1000  EOF  \\
KKMC    ->  1  5  CL5  1000  EOF  \\
```

{node name -> day to start consuming, day to end consuming, class of supply, amount to consume}

VIII. VEHFLD.DAT

```
5
TRUCKS ->
     M35A2  CARGO 35 50 0 500 2000 15 12 20 8 1920 10000
          .044  40.4  750
     M54A1  CARGO 40 75 0 550 3500 15 12 25 8 2400 10000
          .041  40.4  753
     LMTV   CARGO 35 50 0 500 2000 15 12 20 8 1920 10000
          .008  40.4  2200
     M939   CARGO 35 50 0 500 2000 15 12 20 8 1920 10000
          .008  40.4  1800
     HEMTT  CARGO 35 50 0 500 2000 15 12 20 8 1920 10000
          .005  34.0  2675   \\
```

```
TRAC_TRLS ->
      Tractor TRAILER 50 50 0 500 2000 15 12 20 8 1920 5000
            .01   40.0  1000
      M939T TRAILER  50 50 0 500 2000 15 12 20 8 1920 5000
            .008  40.4  2500
      M915T TRAILER  50 50 0 500 2000 15 12 20 8 1920 5000
            .09   34.0  1160 \\
TRAC_LOWBOYS ->
      Trailer  CARGO 50 50 0 500 2000 15 12 20 8 1920 5000
            .01   40.0  500  \\
HETTS ->
      Hett      HETT 70 50 0 500 5000 20 0 403 0 0 140000
            .005  34.0  2500
      M911H     HETT 70 50 0 500 5000 20 0 317 0 0 140000
            .009  34.0  1630
      M1070H    HETT 70 50 0 500 5000 20 0 403 0 0 140000
            .005  34.0  2500  \\
WRECKERS ->
      Wrecker RECOVERY 40 75 0 550 3600 15 0 0 0 0 20000
            .017  40.4  1700  \\

{type vehicle -> model, type, fuelCap, fuelConsump, odometer, engHrs, assetWeight,
      assetLength, height, length, width, cubeFt, weight
      maintManHrs, adminLogTime, MMBF}
```

IX. TRAVELR.DAT

```
10
M35A2 ->
      55.0 48.7 48.7 32.4 40.0 39.6 39.6 29.6
      24.9 24.9 24.9 20.2 18.0 18.0 18.0 17.5  EOF  \\
Wrecker ->
      40.0 39.6 39.6 28.3 40.0 37.9 37.9 28.1
      24.8 24.8 24.8 20.1 24.8 24.8 24.8 20.1  EOF  \\
Hett ->
      30.9 16.6 16.6  8.1 27.0 16.1 16.1  7.8
       8.2  8.1  8.1  5.4  8.2  7.7  7.7  5.2  EOF \\
LMTV ->
      40.0 39.6 39.6 28.3 40.0 37.9 37.9 28.1
      24.8 24.8 24.8 20.1 24.8 24.8 24.8 20.1  EOF  \\
M939 ->
      40.0 35.7 35.7 22.7 40.0 34.0 34.0 22.5
      23.0 21.8 21.8 15.8 10.7 10.7 10.7 10.7  EOF  \\
```

```
M939T ->
      34.4 20.1 20.1 10.3 31.8 19.3 19.3 10.2
      13.7 10.9 10.9  7.2 10.7  8.   8.6  6.1  EOF  \\
HEMTT ->
      38.0 38.0 38.0 25.3 38.0 34.4 34.4 24.4
      24.7 23.3 23.3 18.0 16.9 16.9 16.9 15.4  EOF  \\
M915T ->
      23.0 21.2 21.2 12.5 23.0 20.5 20.5 12.3
      14.0 11.2 11.2  8.3 10.0  9.7  9.7  7.5  EOF  \\
M911H ->
      21.0 13.4 13.4  7.7 19.3 11.8 11.8  7.4
       8.6  7.6  7.6  4.5  7.9  6.8  6.8  4.2  EOF  \\
M1070H ->
      30.9 16.6 16.6  8.1 27.0 16.1 16.1  7.8
       8.2  8.1  8.1  5.4  8.2  7.7  7.7  5.2  EOF \\

{vehicle ->
      concrete
        flat rollingHills hillsCurves mountainous
      bituminous
        flat rollingHills hillsCurves mountainous
      gravel
        flat rollingHills hillsCurves mountainous
      dirt
        flat rollingHills hillsCurves mountainous}
```

X. VEHOWN.DAT

```
2
Dammam ->      TRUCKS       none
               TRAC_TRLS    none
               TRAC_LOWBOYS none
               HETTS        Hett   385
               WRECKERS     Wrecker  4  \\
KKMC ->        TRUCKS       none
               TRAC_TRLS    none
               TRAC_LOWBOYS none
               HETTS        none
               WRECKERS     none  \\

{node ->  type vehicle, model, number of}
```

XI. LINKS.DAT

```
9
Economy -> Dammam   20.0  concrete  flat  \\
Dammam  -> AlJubayl 60.0  concrete  flat
        CSCy    113.0  concrete  flat  \\
AlJubayl -> Bastogne 72.0  concrete  flat
        Dammam   60.0  concrete  flat  \\
Bastogne -> AlJubayl 72.0  concrete  flat
        ALPHA   101.0  concrete  flat  \\
ALPHA   -> Bastogne 101.0  concrete  flat
        KKMC    101.0  concrete  flat  \\
KKMC    -> ALPHA   101.0  concrete  flat  \\
CSCy    -> Dammam  113.0  concrete  flat
        Riyadh  113.0  concrete  flat  \\
Riyadh  -> CSCy    113.0  concrete  flat
        DELTA   175.0  concrete  flat  \\
DELTA   -> Riyadh  175.0  concrete  flat
        KKMC2   127.0  concrete  flat  \\
```

{link origin -> link destination distance, roadSurface, terrain}

XII. ROUTES.DAT

```
9
Economy ->  Economy Dammam  EOF  \\
Dammam  ->
     Dammam AlJubayl Bastogne ALPHA KKMC
     Dammam AlJubayl Bastogne ALPHA
     Dammam AlJubayl Bastogne
     Dammam AlJubayl
     Dammam CSCy Riyadh DELTA KKMC2
     Dammam CSCy Riyadh DELTA
     Dammam CSCy Riyadh
     Dammam CSCy  EOF  \\
AlJubayl ->
     AlJubayl Bastogne  EOF  \\
Bastogne ->
     Bastogne ALPHA  EOF  \\
ALPHA ->
```

```
      ALPHA KKMC  EOF  \\
KKMC ->
      KKMC ALPHA Bastogne AlJubayl Dammam
      KKMC ALPHA  EOF  \\
KKMC2 ->
      KKMC2 DELTA Riyadh CSCy Dammam  EOF  \\
AlJubayl ->
      AlJubayl Dammam
      AlJubayl Bastogne
      AlJubayl Bastogne ALPHA
      AlJubayl Bastogne ALPHA KKMC
      AlJubayl Bastogne ALPHA ECHO
      AlJubayl Bastogne ALPHA ECHO CHARLIE  EOF \\
Dhahran ->
      Dhahran CSCy
      Dhahran CSCy Riyadh
      Dhahran CSCy Riyadh DELTA
      Dhahran CSCy Riyadh DELTA KKMC  EOF \\
```

{node -> start node, next node...next to last, end node}

XIII. MISSION.DAT

```
1
Dammam -> 116    M1  120251   312  HETT  0  TRUE  KKMC 10
          24    M88  112100   325  HETT  0  TRUE  KKMC 10
          12   AVLB   91999   340  HETT  0  TRUE  KKMC 10
           6    CEV  111198   351  HETT  0  TRUE  KKMC 10
          64     M2   41899   254  HETT  0  TRUE  KKMC 10
          18     M3   41500   254  HETT  0  TRUE  KKMC 10
          12   M901   26001   191  HETT  0  TRUE  KKMC 10
          29   M577   23953   191  HETT  0  TRUE  KKMC 10
           3   M578   53572   220  HETT  0  TRUE  KKMC 10
          30   M548   16000   232  HETT  0  TRUE  KKMC 10
          16 FIST-V   26001   191  HETT  0  TRUE  KKMC 10
          27    ACE   26592   191  HETT  0  TRUE  KKMC 10
           6   M106   19852   194  HETT  0  TRUE  KKMC 10
         118   M113   23442   191  HETT  0  TRUE  KKMC 10
          24   M109   46540   244  HETT  0  TRUE  KKMC 10  EOF \\
```

{node -> number to create, name of item, weight in lbs,
length in inches, method of resupply, priority, nonseparable,destination, time to begin mission}

XIV. SEEDS.DAT

This data file (not shown) contains an integer on the first line that represents the number of seeds in the file. The remaining numbers are random seeds that are generated by MODSIM II™'s random number seed generated.

APPENDIX D - ITTSS PROGRAM CODE

ITTSS was entirely written on a 486DX-25mhz personal computer. MODSIM II™ can be run on a variety of operating systems. This simulation model was written with the PC-OS/2 version that requires the PC-OS/2 version 1.1 or later and the C compiler, Microsoft C 5.0. Programs are interchangeable on each system as long as the particular version contains the required libraries the model utilizes.

ITTSS was written on a personal computer to insure that the model did not need the support of a mainframe or a large database. The average compilation time was 10 to 20 minutes depending on how many of the modules required recompilation. Run times were between 1 and 4 minutes, depending on the setup of the problem being run to include number of assets created and number of cargo to be moved.

```
MAIN MODULE ITTSS;
  FROM INPUT IMPORT ReadEmAll;
  FROM Debug IMPORT TraceStream;
  FROM CREATET IMPORT CreateTransSystem;
  FROM DEBUGRN IMPORT SetUpD;
  FROM SimMod IMPORT StartSimulation;
  FROM STARTEM IMPORT StartConsuming, StartSupplyActivities,
                      ScheduleMissions, StartReports,
  FROM NETWORK IMPORT NetworkObj;

VAR
        Network : NetworkObj;
        decision : STRING;
BEGIN
SetUpD(TRUE);
ReadEmAll;

CreateTransSystem(Network);

OUTPUT("Do you want to run a COMPLETE Transportation Simulation?");
OUTPUT("Enter  Y  for yes, N for no.");
INPUT(decision);
IF(decision = "Y") OR (decision = "y") OR (decision = "YES") OR (decision = "Yes")

        StartConsuming(Network);

        StartSupplyActivities(Network);

END IF;

ScheduleMissions(Network.NodeList);

StartReports(Network.NodeList);

OUTPUT("--------START SIMULATION---------");

StartSimulation;

OUTPUT("--------FINISHED SIMULATION---------");

END MODULE {ITTS}.

{*********************************************************}

DEFINITION MODULE GLOBAL;
{Contains variables that will be used throughout ITTSS by other objects}
  FROM GrpMod IMPORT QueueObj;

TYPE
        HqNameType = STRING;
        AssetNameType = STRING;
        ModelNameType = STRING;
        NodeNameType = STRING;
        FacilityNameType = STRING;
        TransUnitNameType = STRING;
```

```
        Dimensions = RECORD {Dimensions for stowage of cargo}
    height,
                length,
                width,
                cubeFt,
                loadCap : REAL;
  END RECORD;

        AssetTypeQueue = QueueObj;
        CargoTypeQueue = QueueObj;
        LinkTypeQueue = QueueObj;
        NodeTypeQueue = QueueObj;
        RouteTypeQueue = QueueObj;
        SupplyRecordTypeQueue = QueueObj;
        RequestTypeQueue = QueueObj;
        ConsumerQueue = QueueObj;
       SupplyClassType = STRING;
        DescriptType = STRING;
        DestinationType = STRING;
        CargoType = (AMMO, CONTAINER, FUEL, HETT, WATER, CARGO, RECOVERY);
        AssetStatus = (Motorpool, Committed, Maintenance);
        VehicleType = (TRUCKS, TRACTORS, TRAILERS, TANKERS, WRECKERS);
        roadCharact = (flat, rollingHills, hillsCurves, mountainous, concrete, bituminous, gravel, dirt);
        FailType = ARRAY INTEGER OF REAL;

        PROCEDURE DisposeOfQueue (IN Queue : QueueObj);

END {DEFINITION} MODULE {global}.

IMPLEMENTATION MODULE GLOBAL;
  FROM GrpMod IMPORT QueueObj;

{-----------------------------------------}
PROCEDURE DisposeOfQueue (IN queue : QueueObj);
{-----------------------------------------}
VAR
        dump : ANYOBJ;
BEGIN
        dump := ASK queue First ();
        WHILE dump < > NILOBJ
                DISPOSE(dump);
                dump := ASK queue Next(dump);
        END WHILE;
        DISPOSE(queue);
END PROCEDURE {DisposeOfQueue};

END {IMPLEMENTATION} MODULE {global}.

{************************************************************}

DEFINITION MODULE RGLOBAL;
{All pertinent variables for inputting data are defined here}

CONST
        MasterFileName = "Master.dat";
TYPE
        FileNameType = STRING;
        SArrayType = ARRAY INTEGER OF STRING;
        SHierRecType = RECORD
                TopString : STRING;
                OwnedString : SArrayType;
```

```
        END RECORD;
        SHArrayType = ARRAY INTEGER OF SHierRecType;
        SeedArrayType = ARRAY INTEGER OF INTEGER;
VAR
        NodeSHArray : SHArrayType;
        AssetOwnersSHArray : SHArrayType;
        MotorpoolSHArray : SHArrayType;
        MaintenanceSHArray : SHArrayType;
        SupplySHArray : SHArrayType;
        SupplySourceSHArray : SHArrayType;
        FuelpointSHArray : SHArrayType;
        UnitsSHArray : SHArrayType;
        AssetFieldsSHArray : SHArrayType;
        TravelRatesSHArray : SHArrayType;
        LinkSHArray : SHArrayType;
        RouteSHArray : SHArrayType;
        MissionSHArray : SHArrayType;
        SeedArray : SeedArrayType;
        SeedCount : INTEGER;

END {DEFINITION} MODULE {RGlobal}.

{*******************************************************}

DEFINITION MODULE INPUT;
{This procedure input all data that is required to run ITTSS}

PROCEDURE ReadEmAll;

END {DEFINITION} MODULE {Input}.

IMPLEMENTATION MODULE INPUT;
  FROM IOMod IMPORT StreamObj, FileUseType(Input);
  FROM RGLOBAL IMPORT MasterFileName, NodeSHArray,
        AssetOwnersSHArray, MotorpoolSHArray, MaintenanceSHArray, SupplySHArray,
        SupplySourceSHArray, FuelpointSHArray, UnitsSHArray, AssetFieldsSHArray,
        TravelRatesSHArray, LinkSHArray, RouteSHArray, MissionSHArray, SeedArray, FileNameType;
  FROM READLST IMPORT ReadLst;
  FROM READSED IMPORT ReadTheSeeds;
  FROM Debug IMPORT TraceStream;

VAR
        NodeFileName,
        AssetOwnersFileName,
        MotorpoolFileName,
        MaintenanceFileName,
        SupplyFileName,
        SupplySourceFileName,
        FuelpointFileName,
        UnitsFileName,
        AssetFieldsFileName,
        TravelRatesFileName,
        LinkFileName,
        MissionFileName,
        SeedFileName,
        RouteFileName : FileNameType;

{----------------}
 PROCEDURE ReadNode;
{----------------}
BEGIN
```

```
ReadLst(NodeSHArray , NodeFileName);
END PROCEDURE {ReadNode};

{----------}
 PROCEDURE ReadMotorpool;
{----------}
BEGIN
ReadLst(MotorpoolSHArray, MotorpoolFileName);
END PROCEDURE {ReadMotorpool};

{----------}
 PROCEDURE ReadMaintenance;
{----------}
BEGIN
ReadLst(MaintenanceSHArray, MaintenanceFileName);
END PROCEDURE {ReadMaintenance};

{----------}
 PROCEDURE ReadSupply;
{----------}
BEGIN
ReadLst(SupplySHArray, SupplyFileName);
END PROCEDURE {ReadSupply};

{----------}
 PROCEDURE ReadSupplySource;
{----------}
BEGIN
ReadLst(SupplySourceSHArray, SupplySourceFileName);
END PROCEDURE {ReadSupplySource};

{----------}
 PROCEDURE ReadFuelpoint;
{----------}
BEGIN
ReadLst(FuelpointSHArray, FuelpointFileName);
END PROCEDURE {ReadFuelpoint};

{----------}
 PROCEDURE ReadUnits;
{----------}
BEGIN
ReadLst(UnitsSHArray, UnitsFileName);
END PROCEDURE {ReadUnits};

{----------}
 PROCEDURE ReadAssetFields;
{----------}
BEGIN
ReadLst(AssetFieldsSHArray, AssetFieldsFileName);
END PROCEDURE {ReadAssetFields};

{----------}
 PROCEDURE ReadTravelRates;
{----------}
BEGIN
ReadLst(TravelRatesSHArray, TravelRatesFileName);
END PROCEDURE {ReadTravelRates};

{----------}
 PROCEDURE ReadAssetOwners;
```

```
{-------------------}
BEGIN
ReadLst(AssetOwnersSHArray, AssetOwnersFileName);
END PROCEDURE {ReadAssetOwners};

{-----------------}
 PROCEDURE ReadLinks;
{-----------------}
BEGIN
ReadLst(LinkSHArray , LinkFileName);
END PROCEDURE {ReadLinks};

{-----------------}
 PROCEDURE ReadRoutes;
{-----------------}
BEGIN
ReadLst(RouteSHArray , RouteFileName);
END PROCEDURE {ReadRoutes};

{-----------------}
 PROCEDURE ReadMissions;
{-----------------}
BEGIN
ReadLst(MissionSHArray , MissionFileName);
END PROCEDURE {ReadMissions};

{----------------}
 PROCEDURE ReadEmAll;
{----------------}
VAR
        File : StreamObj;
        str : STRING;
BEGIN
        NEW(File);
        ASK File TO Open(MasterFileName, Input);
        ASK File TO ReadString(NodeFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(MotorpoolFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(SupplyFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(SupplySourceFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(FuelpointFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(UnitsFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(AssetFieldsFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(TravelRatesFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(AssetOwnersFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(LinkFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(RouteFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(MissionFileName);
        ASK File TO ReadLine(str);
        ASK File TO ReadString(SeedFileName);
```

```
ReadNode;
ReadMotorpool;
ReadSupply;
ReadSupplySource;
ReadFuelpoint;
ReadUnits;
ReadAssetFields;
ReadTravelRates;
ReadAssetOwners;
ReadLinks;
ReadRoutes;
ReadMissions;
ReadTheSeeds(SeedFileName);
END PROCEDURE {ReadEmAll};

END {IMPLEMENTATION} MODULE {Input}.

{*****************************************************}

DEFINITION MODULE READLST;
{Integral procedure in the inputting of data to the model}
  FROM RGLOBAL IMPORT SHArrayType, FileNameType;

PROCEDURE ReadLst (INOUT SHArray : SHArrayType;
                   IN FileName : FileNameType);

END {DEFINITION} MODULE {ReadLst}.

IMPLEMENTATION MODULE READLST;
  FROM IOMod IMPORT StreamObj, FileUseType(Input);
  FROM RGLOBAL IMPORT SHArrayType, FileNameType;
  FROM READSH IMPORT ReadSH;
  FROM Debug IMPORT TraceStream;

{-----------------------------------------}
PROCEDURE ReadLst ( INOUT SHArray : SHArrayType;
                    IN FileName : FileNameType);
{-----------------------------------------}
VAR
        File : StreamObj;
        numberOfSH : INTEGER;
        i : INTEGER;
        error : BOOLEAN;
        string : STRING;
BEGIN
NEW(File);
ASK File TO Open(FileName, Input);
ASK File TO ReadInt(numberOfSH);
ASK File TO ReadLine(string);
NEW(SHArray, 1..numberOfSH);
FOR i := 1 TO numberOfSH
        ReadSH(File, SHArray[i], error);
        IF error
                ASK TraceStream TO WriteString("problem reading file " + FileName + " BAD FORMAT DET>");
        END IF;
END FOR;
ASK File TO ObjTerminate();
ASK File TO Delete();
END PROCEDURE {ReadLst};

END {IMPLEMENTATION} MODULE {ReadLst}.
```

```
{************************************************************}

DEFINITION MODULE READSH;
{Integral part in inputting data to the model}
  FROM RGLOBAL IMPORT SHierRecType;
  FROM IOMod IMPORT StreamObj;

PROCEDURE ReadSH ( IN File : StreamObj;
                   OUT SHierRec : SHierRecType;
                   OUT error : BOOLEAN);

END {DEFINITION} MODULE {ReadSH}.

IMPLEMENTATION MODULE READSH;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM IOMod IMPORT StreamObj, FileUseType(Input), ReadKey;
  FROM Debug IMPORT TraceStream;

{----------------------------------------}
 PROCEDURE ReadSH ( IN File : StreamObj;
                   OUT SHierRec : SHierRecType;
                   OUT error : BOOLEAN);
{----------------------------------------}
TYPE
StringRecType = RECORD
        String : STRING;
        Next : StringRecType;
END RECORD;
VAR
        string : STRING;
        numberOfStrings : INTEGER;
        StringRec, OldStringRec : StringRecType;
        first : StringRecType;
        arrow : STRING;
        stringRec : StringRecType;
        i : INTEGER;
        z : CHAR;
BEGIN
NEW(SHierRec);
ASK File TO ReadString(SHierRec.TopString);
NEW(StringRec);
numberOfStrings := 1;
first := StringRec;
ASK File TO ReadString(arrow);
IF arrow < > "->"
        ASK TraceStream TO WriteString("file not formatted correctly");
        error := TRUE;
        RETURN;
ELSE
        error := FALSE;
END IF;
WHILE string < > "\\"
        ASK File TO ReadString(string);
        IF string = ".."
                ASK File TO ReadLine(string);
        ELSE
                OldStringRec := StringRec;
                StringRec.String := string;
                NEW(StringRec);
                OldStringRec.Next := StringRec;
                numberOfStrings := numberOfStrings + 1;
```

```
        END IF;
END WHILE;
ASK File TO ReadLine(string);
IF (numberOfStrings > 0) AND NOT error
        NEW(SHierRec.OwnedString, 1..numberOfStrings - 2);
        stringRec := first;
        FOR i := 1 TO numberOfStrings - 2
                SHierRec.OwnedString[i] := stringRec.String;
                stringRec := stringRec.Next;
        END FOR;
END IF;
END PROCEDURE {ReadSH};

END {IMPLEMENTATION} MODULE {ReadSH}.

{*********************************************************}

DEFINITION MODULE FINDSH;
{When given a name, it searchs a data array and gives back the correct record requested.  Used for giving objects the initial values
for their fields}
  FROM RGLOBAL IMPORT SHierRecType,SHArrayType;

PROCEDURE FindSHRec (IN SHArray : SHArrayType;
                     IN TopString : STRING;
                     OUT SHRec : SHierRecType);

END {DEFINITION} MODULE {FindSH}.

IMPLEMENTATION MODULE FINDSH;
  FROM RGLOBAL IMPORT SHierRecType,SHArrayType;
  FROM Debug IMPORT TraceStream;

{-----------------------------------------}
 PROCEDURE FindSHRec (IN SHArray : SHArrayType;
                      IN TopString : STRING;
                      OUT SHRec : SHierRecType);
{-----------------------------------------}
VAR
        ThisRec : SHierRecType;
        i : INTEGER;
BEGIN
NEW(SHRec);
i := 0;
REPEAT
        INC(i);
        ThisRec := SHArray[i];
UNTIL ((i >= HIGH(SHArray)) OR (ThisRec.TopString = TopString));
IF (ThisRec.TopString = TopString)
        SHRec := ThisRec;
ELSE
        ASK TraceStream TO WriteString("SHRec is a NILREC!");
        SHRec := NILREC;
END IF;
END PROCEDURE {FindSHRec};

END {IMPLEMENTATION} MODULE {FindSH}.

{*********************************************************}

DEFINITION MODULE READSED;
{Reads the seed from a seed file.  Seeds are used for the mean miles between operational failures for each asset}
```

```
  FROM RGLOBAL IMPORT FileNameType;

PROCEDURE ReadSeed () : INTEGER;
PROCEDURE ReadTheSeeds (IN FileName : FileNameType);

END {DEFINITION} MODULE {ReadSed}.

IMPLEMENTATION MODULE READSED;
  FROM Debug IMPORT TraceStream;
  FROM IOMod IMPORT FileUseType(Input), StreamObj;
  FROM RGLOBAL IMPORT FileNameType, SeedCount, SeedArray;

{-------------------------}
 PROCEDURE ReadSeed () : INTEGER;
{-------------------------}
BEGIN
IF (SeedCount > HIGH(SeedArray))
        ASK TraceStream TO WriteString("Ran out of seeds ");
        ASK TraceStream TO WriteLn;
        HALT;
        RETURN(0);
ELSE
        IF (SeedCount < = 0)
                SeedCount := 1;
        END IF;
        INC(SeedCount);
        RETURN(SeedArray[SeedCount - 1]);
END IF;
END PROCEDURE {ReadSeed};

{-----------------------------------------------}
 PROCEDURE ReadTheSeeds (IN FileName : FileNameType);
{-----------------------------------------------}
VAR
        file : StreamObj;
        str : STRING;
        i : INTEGER;
        NumberOfSeeds : INTEGER;
BEGIN
NEW(file);
ASK file TO Open(FileName, Input);
ASK file TO ReadInt(NumberOfSeeds);
NEW(SeedArray, 1..NumberOfSeeds);
FOR i := 1 TO NumberOfSeeds
        ASK file TO ReadInt(SeedArray[i]);
        ASK file TO ReadLine(str);
END FOR;
END PROCEDURE {ReadTheSeeds};

END {IMPLEMENTATION} MODULE {ReadSed}.

{*****************************************************}

DEFINITION MODULE DEBUGRN;
{Sets up a tracestream that assists in debugging any compiling or runtime errors.  If SetUpD in the main module is true, the trace
is on, if false, only errors will be written to the debug.out file}

PROCEDURE SetUpD (IN TraceOn : BOOLEAN);

END {DEFINITION} MODULE {DebugRn}.
```

```
IMPLEMENTATION MODULE DEBUGRN;
  FROM IOMod IMPORT FileUseType(Output);
  FROM Debug IMPORT TraceStream;
  FROM UtilMod IMPORT DateTime;
{-----------------------------------}
 PROCEDURE SetUpD (IN TraceOn : BOOLEAN);
{-----------------------------------}
VAR
        DT : STRING;
BEGIN
NEW(TraceStream);
ASK TraceStream TO Open ("debug.out", Output);
DateTime(DT);
ASK TraceStream TO WriteString(DT);
ASK TraceStream TO WriteLn;
ASK TraceStream TO WriteLn;
ASK TraceStream TO WriteLn;
IF TraceOn
        ASK TraceStream TO TraceOn;
        OUTPUT("-----TRACE ON---------");
        ASK TraceStream TO WriteString("Initially, trace is on.");
        ASK TraceStream TO WriteLn;
ELSE
        ASK TraceStream TO TraceOff;
        ASK TraceStream TO WriteString("Initially, trace is off.");
        ASK TraceStream TO WriteLn;
END IF;
END PROCEDURE {SetUpD};

END {IMPLEMENTATION} MODULE {DebugRn}.

{*********************************************************}

DEFINITION MODULE NETWORK;
{Network is the top of the tree that contains all nodes, links and routes.  All access to these objects are through the network's queue
of these objects}
  FROM GLOBAL IMPORT NodeTypeQueue,LinkTypeQueue,RouteTypeQueue;

TYPE
NetworkObj = OBJECT
        NodeList : NodeTypeQueue;
        RouteList : RouteTypeQueue;
        LinkList : LinkTypeQueue;

        ASK METHOD ObjInit;
END OBJECT;

END {DEFINITION} MODULE {network}.

IMPLEMENTATION MODULE NETWORK;
  FROM GLOBAL IMPORT NodeTypeQueue,LinkTypeQueue,RouteTypeQueue;

OBJECT NetworkObj;
{----------------}
 ASK METHOD ObjInit;
{----------------}
BEGIN
        NEW(NodeList);
        NEW(RouteList);
        NEW(LinkList);
END METHOD {ObjInit};
```

```
END OBJECT {NetworkObj};

END {IMPLEMENTATION} MODULE {network}.

{*****************************************************************}

DEFINITION MODULE CREATET;
{These are general procedures that initialize the transportation system for ITTSS.  Nodes, links, routes, and all objects within each
node are created}

  FROM RGLOBAL IMPORT SHierRecType;
  FROM NODE IMPORT NodeObj;
  FROM NETWORK IMPORT NetworkObj;

 PROCEDURE CreateTransSystem(OUT Network : NetworkObj);
 PROCEDURE CreateNetwork (IN Network : NetworkObj);
 PROCEDURE CreateAttachedUnits (IN Node : NodeObj;
                                IN NodeSHRec : SHierRecType);

END {DEFINITION} MODULE {createt}.

IMPLEMENTATION MODULE CREATET;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT RouteTypeQueue, NodeTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType, NodeSHArray, MaintenanceSHArray, MotorpoolSHArray,
         SupplySHArray, FuelpointSHArray;
  FROM NETWORK IMPORT NetworkObj;
  FROM NODE IMPORT NodeObj;
  FROM FINDSH IMPORT FindSHRec;
  FROM ROUTE IMPORT CreateLinks, CreateRoutes;
  FROM MAINT IMPORT MaintenanceObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM ASSET IMPORT AssetObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM FUELPT IMPORT FuelpointObj;
  FROM UNITS IMPORT UnitsObj;
  FROM CREATE IMPORT CreateMotorpool, CreateMaintenance, CreateSupply,
        CreateFuelpoint, CreateUnits, CreateSupplySources;

{-------------------------------------------}
 PROCEDURE CreateTransSystem(OUT Network : NetworkObj);
{-------------------------------------------}
VAR
        i : INTEGER;
        NodeSHRec : SHierRecType;
BEGIN
NEW(Network);
CreateNetwork (Network);
END PROCEDURE {CreateTransSystem};

{-----------------------------------------}
 PROCEDURE CreateNetwork (IN Network : NetworkObj);
{-----------------------------------------}
VAR
        i : INTEGER;
        Node : NodeObj;
        NodeSHRec : SHierRecType;
        MasterRouteList : RouteTypeQueue;
        MasterNodeList : NodeTypeQueue;
BEGIN
```

```
{------------------------Creating Nodes-----------------}
FOR i := 1 TO HIGH(NodeSHArray)
        NEW(Node);
        ASK Node TO GetName(NodeSHArray[i].TopString);
        ASK Node TO GetClearances;
        FindSHRec(NodeSHArray, ASK Node name, NodeSHRec);
        ASK Node TO GetRestrictions(NodeSHRec);
        CreateAttachedUnits(Node, NodeSHRec);
        ASK Network.NodeList TO Add (Node);
        CreateLinks(ASK Node name, Network.LinkList);
END FOR;
NEW(MasterRouteList);
FOR i := 1 TO HIGH(NodeSHArray)
        CreateRoutes(NodeSHArray[i].TopString, Network, MasterRouteList);
END FOR;
{Give each Motorpool a copy of the Master Lists}
NEW(MasterNodeList);
MasterNodeList := Network.NodeList;
Node := ASK Network.NodeList First();
REPEAT
        IF(Node.myMotorpool < > NILOBJ)
                ASK Node.myMotorpool TO GetMasterLists (MasterRouteList, MasterNodeList);
        END IF;
        Node := ASK Network.NodeList Next(Node);
UNTIL (Node = NILOBJ);

Node := ASK Network.NodeList First();
REPEAT
        IF(Node.mySupply < > NILOBJ)
                CreateSupplySources(Node.mySupply,Network);
        END IF;
        Node := ASK Network.NodeList Next(Node);
UNTIL (Node = NILOBJ);
END PROCEDURE {CreateNetwork};

{-----------------------------------------------------}
 PROCEDURE CreateAttachedUnits (IN Node : NodeObj;
                                IN NodeSHRec : SHierRecType);
{-----------------------------------------------------}
VAR
        i : INTEGER;
        Motorpool : MotorpoolObj;
        Maintenance : MaintenanceObj;
        Supply : SupplyObj;
        Fuelpoint : FuelpointObj;
        Units : UnitsObj;
        MotorpoolName, MaintenanceName, SupplyName, FuelpointName ,      UnitsName : STRING;
BEGIN
IF (NodeSHRec = NILREC)
        FindSHRec (NodeSHArray, ASK Node name, NodeSHRec);
END IF;
CreateMotorpool(ASK Node name, NodeSHRec, Motorpool);
ASK Node TO GetMotorpool(Motorpool);

CreateMaintenance(ASK Node name, NodeSHRec, Maintenance);
IF((Motorpool < > NILOBJ) AND (Maintenance < > NILOBJ))
        ASK Maintenance TO GetMyMotorpool(Motorpool);
END IF;
ASK Node TO GetMaintenance(Maintenance);

CreateSupply(ASK Node name, NodeSHRec, Supply);
```

```
ASK Node TO GetSupply(Supply);

CreateFuelpoint(ASK Node name, NodeSHRec, Fuelpoint);
ASK Node TO GetFuelpoint(Fuelpoint);

CreateUnits(ASK Node name, NodeSHRec, Units);
ASK Node TO GetUnits(Units);

IF(Motorpool < > NILOBJ)
        ASK Supply TO GetMyMotorpool(Motorpool);
END IF;

IF(Supply < > NILOBJ)
        ASK Units TO GetMySupply(Supply);
END IF;
END PROCEDURE {CreateAttachedUnits};

END {IMPLEMENTATION} MODULE {createt}.

{*****************************************************}

DEFINITION MODULE CREATE;
{Each node's motorpools, maintenance facilities, supply points, fuelpoints, units, and each supply point's supply source are created
here}
  FROM RGLOBAL IMPORT SHierRecType;
  FROM NODE IMPORT NodeObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM MAINT IMPORT MaintenanceObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM FUELPT IMPORT FuelpointObj;
  FROM UNITS IMPORT UnitsObj;
  FROM NETWORK IMPORT NetworkObj;

 PROCEDURE CreateMotorpool (IN name : STRING;
                            IN NodeSHRec : SHierRecType;
                            OUT Motorpool : MotorpoolObj);
 PROCEDURE CreateMaintenance (IN name : STRING;
                              IN NodeSHRec : SHierRecType;
                              OUT Maintenance : MaintenanceObj);
 PROCEDURE CreateSupply (IN name : STRING;
                         IN NodeSHRec : SHierRecType;
                         OUT Supply : SupplyObj);
 PROCEDURE CreateFuelpoint (IN name : STRING;
                            IN NodeSHRec : SHierRecType;
                            OUT Fuelpoint : FuelpointObj);
 PROCEDURE CreateUnits (IN name : STRING;
                        IN NodeSHRec : SHierRecType;
                        OUT Units : UnitsObj);

 PROCEDURE CreateSupplySources (IN Supply : SupplyObj;
                                IN Network : NetworkObj);

END {DEFINITION} MODULE {create}.

IMPLEMENTATION MODULE CREATE;
  FROM Debug IMPORT TraceStream;
  FROM RGLOBAL IMPORT SHierRecType, NodeSHArray, MotorpoolSHArray, MaintenanceSHArray, SupplySHArray,
        FuelpointSHArray, UnitsSHArray, SupplySourceSHArray;
  FROM NODE IMPORT NodeObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM MAINT IMPORT MaintenanceObj;
```

```
  FROM SUPPLY IMPORT SupplyObj;
  FROM FUELPT IMPORT FuelpointObj;
  FROM UNITS IMPORT UnitsObj;
  FROM FINDSH IMPORT FindSHRec;
  FROM NETWORK IMPORT NetworkObj;

{-------------------------------------------------}
 PROCEDURE CreateMotorpool (IN name : STRING;
                            IN NodeSHRec : SHierRecType;
                            OUT Motorpool : MotorpoolObj);
{-------------------------------------------------}
VAR
        i : INTEGER;
        MotorpoolName : STRING;
BEGIN
i := 1;
WHILE ((NodeSHRec.OwnedString[i] < > "M") AND
        (NodeSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
REPEAT
        MotorpoolName := NodeSHRec.OwnedString[i];
        NEW(Motorpool);
        ASK Motorpool TO ObjInit;
        ASK Motorpool TO GetName (MotorpoolName);
        ASK Motorpool TO GetFields(name);
        ASK Motorpool TO GetAssets (name);
        INC(i);
UNTIL ((i > HIGH(NodeSHRec.OwnedString)) OR
        (NodeSHRec.OwnedString[i] = "\\") OR
        (NodeSHRec.OwnedString[i] = "MA"));
END PROCEDURE {CreateMotorpool};

{-------------------------------------------------}
 PROCEDURE CreateMaintenance (IN name : STRING;
                            IN NodeSHRec : SHierRecType;
                              OUT Maintenance : MaintenanceObj);
{-------------------------------------------------}
VAR
        i : INTEGER;
        MaintenanceName : STRING;
BEGIN
i := 1;
WHILE ((NodeSHRec.OwnedString[i] < > "MA") AND
        (NodeSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
REPEAT
        MaintenanceName := NodeSHRec.OwnedString[i];
        NEW(Maintenance);
        ASK Maintenance TO GetName (MaintenanceName);
        {ASK Maintenance TO GetFields(name);}
        INC(i);
UNTIL ((i > HIGH(NodeSHRec.OwnedString)) OR
        (NodeSHRec.OwnedString[i] = "\\") OR
        (NodeSHRec.OwnedString[i] = "S"));
END PROCEDURE {CreateMaintenance};

{-------------------------------------------------}
```

```
 PROCEDURE CreateSupply (IN name : STRING;
                         IN NodeSHRec : SHierRecType;
                         OUT Supply : SupplyObj);
{-------------------------------------------}
VAR
        i : INTEGER;
        SupplyName : STRING;
BEGIN
i := 1;
WHILE ((NodeSHRec.OwnedString[i] < > "S") AND
        (NodeSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
REPEAT
        SupplyName := NodeSHRec.OwnedString[i];
        NEW(Supply);
        ASK Supply TO GetName (SupplyName);
        ASK Supply TO GetFields(name);
        INC(i);
UNTIL ((i > HIGH(NodeSHRec.OwnedString)) OR
        (NodeSHRec.OwnedString[i] = "\\") OR
        (NodeSHRec.OwnedString[i] = "F"));
END PROCEDURE {CreateSupply};

{-------------------------------------------------}
 PROCEDURE CreateFuelpoint (IN name : STRING;
                            IN NodeSHRec : SHierRecType;
                            OUT Fuelpoint : FuelpointObj);
{-------------------------------------------------}
VAR
        i : INTEGER;
        FuelpointName : STRING;
BEGIN
i := 1;
WHILE ((NodeSHRec.OwnedString[i] < > "F") AND
        (NodeSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
REPEAT
        FuelpointName := NodeSHRec.OwnedString[i];
        NEW(Fuelpoint);
        ASK Fuelpoint TO GetName (FuelpointName);
        ASK Fuelpoint TO GetFields(name);
        INC(i);
UNTIL ((i > HIGH(NodeSHRec.OwnedString)) OR
        (NodeSHRec.OwnedString[i] = "\\") OR
        (NodeSHRec.OwnedString[i] = "U"));
END PROCEDURE {CreateFuelpoint};

{-----------------------------------------}
 PROCEDURE CreateUnits (IN name : STRING;
                        IN NodeSHRec : SHierRecType;
                        OUT Units : UnitsObj);
{-------------------------------------------}
VAR
        i : INTEGER;
        UnitsName : STRING;
BEGIN
i := 1;
```

```
WHILE ((NodeSHRec.OwnedString[i] < > "U") AND
        (NodeSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
REPEAT
        UnitsName := NodeSHRec.OwnedString[i];
        NEW(Units);
        ASK Units TO GetName (UnitsName);
        ASK Units TO GetFields(name);
        INC(i);
UNTIL ((i > HIGH(NodeSHRec.OwnedString)) OR
        (NodeSHRec.OwnedString[i] = "\\") OR
        (NodeSHRec.OwnedString[i] = "EOF"));
END PROCEDURE {CreateUnits};

{-------------------------------------------------}
 PROCEDURE CreateSupplySources (IN Supply : SupplyObj;
                                IN Network : NetworkObj);
{-------------------------------------------------}
VAR
        i : INTEGER;
        source : STRING;
        Node : NodeObj;
        SupplySourceSHRec : SHierRecType;
        sourceFound : BOOLEAN;
BEGIN
sourceFound := FALSE;
FindSHRec(SupplySourceSHArray, ASK Supply name, SupplySourceSHRec);
source := SupplySourceSHRec.OwnedString[1];
{Find the source SupplyObj that belongs to the node}
Node := ASK Network.NodeList First();
REPEAT
        IF(source = Node.mySupply.name)
                ASK Supply TO GetSupplySource (Node.mySupply);
                sourceFound := TRUE;
        ELSE
                Node := ASK Network.NodeList Next(Node);
        END IF;
UNTIL ((Node=NILOBJ) OR (sourceFound));
END PROCEDURE {CreateSupplySources};

END {IMPLEMENTATION} MODULE {create}.

{*****************************************************}

DEFINITION MODULE CREATEA;
{This procedure creates each asset for a motorpool, by giving it a name and all fields that necessary}
  FROM ASSET IMPORT AssetObj;

PROCEDURE CreateAsset (IN VehicleType : STRING;
                       IN model : STRING;
                       IN i : INTEGER;
                       IN Unit : STRING;
                       INOUT asset : AssetObj);

END {DEFINITION} MODULE {createa}.

IMPLEMENTATION MODULE CREATEA;
  FROM Debug IMPORT TraceStream;
  FROM RGLOBAL IMPORT SHierRecType, AssetFieldsSHArray;
```

```
  FROM ASSET IMPORT AssetObj;
  FROM FINDSH IMPORT FindSHRec;

{-------------------------------------}
 PROCEDURE CreateAsset (IN VehicleType : STRING;
                        IN model : STRING;
                        IN k : INTEGER;
                        IN Unit : STRING;
                        INOUT Asset : AssetObj);
{-------------------------------------}
VAR
        i : INTEGER;
        AssetFieldsSHRec : SHierRecType;
BEGIN
FindSHRec(AssetFieldsSHArray, VehicleType, AssetFieldsSHRec);
i := 1;
WHILE ((AssetFieldsSHRec.OwnedString[i] < > model) AND
        (AssetFieldsSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
ASK Asset TO GetName(model, k, Unit);
ASK Asset TO GetFields(AssetFieldsSHRec);
END PROCEDURE {CreateAsset};

END {IMPLEMENTATION} MODULE {createa}.

{*********************************************************}

DEFINITION MODULE NODE;
{Fields and methods of a node are defined here.  A procedure that finds a node given its name is included}
  FROM GrpMod IMPORT QueueObj;
  FROM ResMod IMPORT ResourceObj;
  FROM GLOBAL IMPORT NodeNameType,CargoTypeQueue,AssetTypeQueue,  NodeTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM MAINT IMPORT MaintenanceObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM FUELPT IMPORT FuelpointObj;
  FROM UNITS IMPORT UnitsObj;
EXPORTTYPE
        NodeObj = OBJECT; FORWARD;
TYPE
Clearance = ResourceObj;
DeadlinedQueue = QueueObj;
DeadlinePointObj = OBJECT
        recoverer : STRING;
        AssetQueue : AssetTypeQueue;
        ASK METHOD ObjInit;
        ASK METHOD GetRecoverer (IN name : STRING);
END OBJECT;

NodeObj = OBJECT
        name : NodeNameType;
        fuelLevel : REAL;
        fuelCap : REAL;
        fuelReserve : REAL;
        loadTime : REAL;
        unloadTime : REAL;
        myMotorpool : MotorpoolObj;
        myMaintenance : MaintenanceObj;
        mySupply : SupplyObj;
```

```
        myFuelpoint : FuelpointObj;
        myUnits : UnitsObj;
        deadlinePointQueue : DeadlinedQueue; {location of any NMC assets}
        clearance : Clearance;     {authority and right of passage thru node}
        maxMilesAllowed : REAL;       {at destin, miles before mandatory standwn}
        maxOpHoursAllowed : REAL;     {same, but in operating hours}
        standDownTime : REAL;         {hours standown before convoy can leave}
        milesBeforeBreak : REAL;      {any node, cept destin, max miles before bk}
        breakTime : REAL;         {hours for breaktime, before proceeding}
        dayToStart : INTEGER;  {day activities start at node}
        dayToEnd : INTEGER;    {day activities end at node}

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        ASK METHOD GetRestrictions (IN NodeSHRec : SHierRecType);
        ASK METHOD GetMotorpool (IN Motorpool : MotorpoolObj);
        ASK METHOD GetMaintenance (IN Maintenance : MaintenanceObj);
        ASK METHOD GetSupply (IN Supply : SupplyObj);
        ASK METHOD GetFuelpoint (IN Fuelpoint : FuelpointObj);
        ASK METHOD GetUnits (IN Units : UnitsObj);
        ASK METHOD GetClearances;

END OBJECT {node};
{-----------------------------------------}
 PROCEDURE FindNode (IN nodeName : STRING;
                     IN NodeList : NodeTypeQueue;
                     OUT Node : NodeObj);
{-----------------------------------------}

END {DEFINITION} MODULE {node}.

IMPLEMENTATION MODULE NODE;
  FROM GLOBAL IMPORT NodeNameType,CargoTypeQueue,NodeTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM CARGO IMPORT CargoObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM MAINT IMPORT MaintenanceObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM FUELPT IMPORT FuelpointObj;
  FROM UNITS IMPORT UnitsObj;
  FROM Debug IMPORT TraceStream;

OBJECT DeadlinePointObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
        NEW(AssetQueue);
END METHOD {ObjInit};

{-----------------------------------}
 ASK METHOD GetRecoverer (IN Name : STRING);
{-----------------------------------}
BEGIN
        recoverer := Name;
END METHOD {GetRecoverer};

END OBJECT {DeadlinePoint};

OBJECT NodeObj;
{-----------------}
```

```
 ASK METHOD ObjInit;
{-----------------}
BEGIN
        NEW(deadlinePointQueue);
        NEW(clearance);
END METHOD {ObjInit};

{-----------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{-----------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{-----------------------------------------------}
 ASK METHOD GetRestrictions (IN NodeSHRec : SHierRecType);
{-----------------------------------------------}
BEGIN
        maxMilesAllowed := STRTOREAL(NodeSHRec.OwnedString[1]);
        maxOpHoursAllowed := STRTOREAL (NodeSHRec.OwnedString[2]);
        standDownTime := STRTOREAL (NodeSHRec.OwnedString[3]);
        milesBeforeBreak := STRTOREAL (NodeSHRec.OwnedString[4]);
        breakTime := STRTOREAL (NodeSHRec.OwnedString[5]);
        dayToStart := STRTOINT (NodeSHRec.OwnedString[6]);
        dayToEnd := STRTOINT (NodeSHRec.OwnedString[7]);
END METHOD {GetRestrictions};

{-----------------------------------------------}
 ASK METHOD GetMotorpool (IN Motorpool : MotorpoolObj);
{-----------------------------------------------}
BEGIN
myMotorpool := Motorpool;
END METHOD {GetMotorpool};

{-------------------------------------------------}
 ASK METHOD GetMaintenance (IN Maintenance : MaintenanceObj);
{-------------------------------------------------}
BEGIN
myMaintenance := Maintenance;
END METHOD {GetMaintenance};

{-----------------------------------}
 ASK METHOD GetSupply (IN Supply : SupplyObj);
{-----------------------------------}
BEGIN
mySupply := Supply;
END METHOD {GetSupply};

{-------------------------------------------}
 ASK METHOD GetFuelpoint (IN Fuelpoint : FuelpointObj);
{-------------------------------------------}
BEGIN
myFuelpoint := Fuelpoint;
END METHOD {GetFuelpoint};

{-----------------------------------}
 ASK METHOD GetUnits (IN Units : UnitsObj);
{-----------------------------------}
BEGIN
myUnits := Units;
END METHOD {GetUnits};
```

```
{----------}
 ASK METHOD GetClearances;
{----------}
BEGIN
        NEW(clearance);
        ASK clearance TO Create(1);
END METHOD {GetClearances};

END OBJECT {Node};
{----------}
 PROCEDURE FindNode (IN nodeName : STRING;
                     IN NodeList : NodeTypeQueue;
                     OUT Node : NodeObj);
{----------}
VAR
        foundNode : BOOLEAN;
BEGIN
foundNode := FALSE;
Node := ASK NodeList First ();
REPEAT
        IF (nodeName = (ASK Node name))
                foundNode := TRUE;
        ELSE
                Node := ASK NodeList Next (Node);
        END IF;
UNTIL ((Node=NILOBJ) OR (foundNode));
END PROCEDURE {FindNode};

END {IMPLEMENTATION} MODULE {node}.

{**********************************************************}

DEFINITION MODULE ROUTE;
{Everything associated with a route are defined here.  Links, routes, creating them and finding them are included.}
  FROM GLOBAL IMPORT NodeNameType,LinkTypeQueue,roadCharact,  RouteTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM NETWORK IMPORT NetworkObj;

TYPE
LinkObj = OBJECT
        origin : NodeNameType;
        destin   : NodeNameType;
        distance : REAL;
        thruput : REAL;
        roadSurface, terrain : roadCharact;

        ASK METHOD GetFields(IN LinkSHRec : SHierRecType;
                             INOUT i : INTEGER);
        ASK METHOD CopyFields(IN link : LinkObj);
END OBJECT {Link};

RouteObj = OBJECT
        origin : NodeNameType;
        destin : NodeNameType;
        LinkRoute : LinkTypeQueue;

        ASK METHOD ObjInit;
        ASK METHOD GetFields(IN origin : STRING;
                             IN destin : STRING);
END OBJECT {Route};
```

```
 PROCEDURE CreateLinks (IN name : STRING;
                        IN LinkList : LinkTypeQueue);
 PROCEDURE FindLink (IN origin : STRING;
                     IN destin : STRING;
                     IN LinkList : LinkTypeQueue;
                     OUT link : LinkObj);
 PROCEDURE CreateRoutes (IN name : STRING;
                         IN Network : NetworkObj;
                         INOUT MasterRouteList : RouteTypeQueue);

END {DEFINITION} MODULE {route}.

IMPLEMENTATION MODULE ROUTE;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT NodeNameType, ALL roadCharact, LinkTypeQueue, RouteTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType, NodeSHArray, LinkSHArray,    RouteSHArray;
  FROM NODE IMPORT NodeObj;
  FROM FINDSH IMPORT FindSHRec;
  FROM NETWORK IMPORT NetworkObj;

OBJECT LinkObj;
{-----------------------------------------------}
 ASK METHOD GetFields(IN LinkSHRec : SHierRecType;
                      INOUT i : INTEGER);
{-----------------------------------------------}
BEGIN
        origin := LinkSHRec.TopString;
        destin := (LinkSHRec.OwnedString[i]);
        INC(i);
        distance := STRTOREAL(LinkSHRec.OwnedString[i]);
        INC(i);
        CASE  (LinkSHRec.OwnedString[i])
                WHEN "concrete":        roadSurface := concrete;
                WHEN "bituminous":      roadSurface := bituminous;
                WHEN "gravel":          roadSurface := gravel;
                WHEN "dirt":            roadSurface := dirt;
                OTHERWISE
                        roadSurface :=concrete;
        END CASE;
        INC(i);
        CASE (LinkSHRec.OwnedString[i])
                WHEN "flat":            terrain := flat;
                WHEN "rollingHills":    terrain := rollingHills;
                WHEN "hillsCurves":     terrain := hillsCurves;
                WHEN "mountainous":terrain := mountainous;
                OTHERWISE
                        terrain := flat;
        END CASE;
END METHOD {GetFields};

{-----------------------------------}
 ASK METHOD CopyFields(IN Link : LinkObj);
{-----------------------------------}
BEGIN
        origin := Link.origin;
        destin := Link.destin;
        distance := Link.distance;
        roadSurface := Link.roadSurface;
        terrain := Link.terrain;
END METHOD {CopyFields};
```

```
END OBJECT {LinkObj};

OBJECT RouteObj;
{----------}
 ASK METHOD ObjInit;
{----------}
BEGIN
        NEW(LinkRoute);
END METHOD {ObjInit};

{----------}
ASK METHOD GetFields(IN start : STRING;
                     IN end : STRING);
{----------}
BEGIN
        origin := start;
        destin := end;
END METHOD {GetFields};

END OBJECT {RouteObj};

{----------}
 PROCEDURE CreateLinks (IN name : STRING;
                        IN LinkList : LinkTypeQueue);
{----------}
VAR
        i : INTEGER;
        Link : LinkObj;
        LinkSHRec : SHierRecType;
BEGIN
i := 1;
FindSHRec(LinkSHArray, name, LinkSHRec);
REPEAT
        NEW(Link);
        ASK Link TO GetFields(LinkSHRec,i);
        ASK LinkList TO Add (Link);
        INC(i);
UNTIL((i > HIGH(LinkSHRec.OwnedString)) OR (LinkSHRec.OwnedString[i] = "\\"));
END PROCEDURE {CreateLinks};

{----------}
 PROCEDURE FindLink (IN start : STRING;
                     IN end : STRING;
                     IN LinkList : LinkTypeQueue;
                     OUT rightLink : LinkObj);
{----------}
VAR
        link : LinkObj;
        linkFound : BOOLEAN;
BEGIN
NEW(rightLink);
link := ASK LinkList First();
REPEAT
        IF((ASK link origin = start) AND (ASK link destin = end))
                ASK rightLink TO CopyFields (link);
                linkFound := TRUE;
        ELSE
                link := ASK LinkList Next(link);
        END IF;
UNTIL((linkFound) OR (link=NILOBJ));
END PROCEDURE {FindLink};
```

```
{-------------------------------------------------------}
 PROCEDURE CreateRoutes (IN name : STRING;
                         IN Network : NetworkObj;
                         INOUT MasterRouteList : RouteTypeQueue);
{-------------------------------------------------------}
VAR
        i : INTEGER;
        link : LinkObj;
        Route : RouteObj;
        RouteSHRec : SHierRecType;
        origin, destin : STRING;
        start, end : STRING;
BEGIN
FindSHRec(RouteSHArray, name, RouteSHRec);
i := 1;
IF (HIGH(RouteSHRec.OwnedString) < 2)
        ASK TraceStream TO WriteString("No routes for node" + name);
ELSE
        REPEAT
                NEW(Route);
                ASK Route TO ObjInit;
                origin := RouteSHRec.OwnedString[i];
                start := origin;
                INC(i);
                end := RouteSHRec.OwnedString[i];
                REPEAT
                        FindLink(start,end,Network.LinkList,link);
                        ASK Route.LinkRoute TO Add (link);
                        start := end;
                        INC(i);
                        end := RouteSHRec.OwnedString[i];
                        IF((end = origin) OR (end = "EOF"))
                                destin := RouteSHRec.OwnedString[i-1];
                                ASK Route TO GetFields(origin,destin);
                        END IF;
                UNTIL((end = origin) OR (end = "EOF"));
                ASK MasterRouteList TO Add (Route);
        UNTIL(end = "EOF");
END IF;
END PROCEDURE {CreateRoutes};

END {IMPLEMENTATION} MODULE {route}.

{*************************************************************}

DEFINITION MODULE ASSET;
{Module contains the methods and fields that describe an asset within ITTSS.  Vehicle objects inherent the methods and fields from
the general asset object.  Specific vehicle objects are further defined}
  FROM GrpMod IMPORT QueueObj;
  FROM GLOBAL IMPORT DescriptType, Dimensions, LinkTypeQueue,  AssetTypeQueue, CargoTypeQueue, ALL roadCharact,
        AssetStatus, FailType;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM CARGO IMPORT CargoObj;
  FROM ROUTE IMPORT LinkObj;
  FROM RandMod IMPORT RandomObj;

EXPORTTYPE
        AssetObj = OBJECT; FORWARD;
TYPE
dimensionsObj = OBJECT          {Dimensions for stowage of cargo}
        height,
```

```
        length,
        width,
        cubeFt,
        weight : REAL;  {vehicle, capacity}

        ASK METHOD GetFields(IN Height : REAL;
                             IN Length : REAL;
                             IN Width : REAL;
                             IN CubeFt : REAL;
                             IN Weight : REAL);
        ASK METHOD AdjustDimensions(IN Height : REAL;
                                    IN Length : REAL;
                                    IN Width : REAL;
                                    IN CubeFt : REAL;
                                    IN Weight : REAL);
        ASK METHOD UpdateLoadWeight(IN weight : REAL);
        ASK METHOD UpdateLengthLoad(IN length : REAL);
        ASK METHOD GetBigValue;
        ASK METHOD GetSmallValue;
END OBJECT {dimensionsObj};

rates = ARRAY roadCharact OF ARRAY roadCharact OF REAL;

AssetObj = OBJECT
        home : STRING;
        MyMotorpool  : STRING;
        model,
        description,
        bumperNumber : DescriptType;
        type : STRING;
        vehType : STRING;
        rateOfTravel : rates;
        fuelCap,
        fuelLevel,              {Current fuel level of vehicle}
        fuelConsump,            {Fuel consumption rate}
        fuelGuage,              {Guage level of fuel, 1/2, etc}
        fuelReserve,
        odometer,
        tripOdometer,
        engHrs,
        assetWeight    : REAL;
        assetLength    : REAL;
        maintManHours : REAL;
        adminLogTime : REAL;
        assetDimens    : dimensionsObj;{Cargo dimensions of vehicle}
        loadDimens : dimensionsObj;
        missionCapable : BOOLEAN;
        tempFix : BOOLEAN;
        status : AssetStatus;
        loadCap : REAL;
        cargoHold : CargoTypeQueue;
        milesToFail : REAL;
        FailureVariateStream : RandomObj;
        MMBF : FailType;

        ASK METHOD ObjInit;
        ASK METHOD ObjTerminate;
        TELL METHOD LoadCargo (IN cargo : CargoObj);
        TELL METHOD UnloadCargo (IN weight : REAL;
                                 IN length : REAL);
        ASK METHOD Refuel (OUT fuelUsed : REAL);
```

```
        ASK METHOD GetBigValue;
        ASK METHOD GetSmallValue;
        ASK METHOD UpdateGuages (IN Link : LinkObj);
        ASK METHOD GetName (IN NewName : DescriptType;
                            IN i : INTEGER;
                            IN Unit : STRING);
        ASK METHOD GetFields (IN AssetFieldsSHRec : SHierRecType);
        ASK METHOD GetFailure(IN AssetFieldsSHRec : SHierRecType);
        ASK METHOD CheckForBreakdown;
        ASK METHOD Break;
        ASK METHOD ResetOdometer;
        ASK METHOD ResetTripOdometer;
        ASK METHOD FixTemporarily;
END OBJECT {asset};

VehicleObj = OBJECT(AssetObj)
END OBJECT {VehicleObj};

TruckObj = OBJECT(VehicleObj)
END OBJECT {TruckObj};

TracTrlObj = OBJECT(VehicleObj)
END OBJECT {TracTrlObj};

TracLowBoyObj = OBJECT(VehicleObj)
END OBJECT {TracLowBoyObj};

HettObj = OBJECT(VehicleObj)
END OBJECT {HettObj};

RecoveryObj = OBJECT(VehicleObj)
        RecoveryLoad : AssetTypeQueue;
        hookTime : REAL;

        TELL METHOD HookUp(IN brokenAsset : AssetObj);
        TELL METHOD UnHook;
        OVERRIDE
                ASK METHOD ObjInit;
END OBJECT {RecoveryObj};

VesselObj = OBJECT(AssetObj)
END OBJECT; {VesselObj}

AirObj = OBJECT(AssetObj)
END OBJECT; {AirObj}

TrainObj = OBJECT(AssetObj)
END OBJECT; {TrainObj}

END {DEFINITION} MODULE {asset}.

IMPLEMENTATION MODULE ASSET;
  FROM GrpMod IMPORT QueueObj;
  FROM GLOBAL IMPORT HqNameType, DescriptType, Dimensions, LinkTypeQueue,
        AssetTypeQueue, CargoTypeQueue, ALL roadCharact, AssetStatus;
  FROM RGLOBAL IMPORT SHierRecType,AssetFieldsSHArray,TravelRatesSHArray;
  FROM FINDSH IMPORT FindSHRec;
  FROM CARGO IMPORT CargoObj;
  FROM ROUTE IMPORT LinkObj;
  FROM Debug IMPORT TraceStream;
  FROM RandMod IMPORT RandomObj;
```

```
 FROM READSED IMPORT ReadSeed;

OBJECT  dimensionsObj;          {Dimensions for stowage of cargo}
{-------------------------------}
 ASK METHOD GetFields(IN Height : REAL;
                      IN Length : REAL;
                      IN Width : REAL;
                      IN CubeFt : REAL;
                      IN Weight : REAL);
{-------------------------------}
BEGIN
height := Height;
length := Length;
width := Width;
cubeFt := CubeFt;
weight := Weight;
END METHOD {GetFields};

{---------------------------------------}
 ASK METHOD AdjustDimensions(IN Height : REAL;
                             IN Length : REAL;
                             IN Width : REAL;
                             IN CubeFt : REAL;
                             IN Weight : REAL);
{---------------------------------------}
BEGIN
END METHOD {AdjustDimensions};

{---------------------------------------}
 ASK METHOD UpdateLoadWeight(IN Weight : REAL);
{---------------------------------------}
BEGIN
        weight := weight + Weight;
END METHOD {UpdateLoadWeight};

{---------------------------------------}
 ASK METHOD UpdateLengthLoad(IN Length : REAL);
{---------------------------------------}
BEGIN
        length := length + Length;
END METHOD {UpdateLoadLength};

{-----------------}
 ASK METHOD GetBigValue;
{-----------------}
BEGIN
        weight := 99999999999999.99;
        length := 99999999999999.99;
END METHOD {GetBigValue};

{-----------------}
 ASK METHOD GetSmallValue;
{-----------------}
BEGIN
        weight := 0.0;
        length := 0.0;
END METHOD {GetSmallValue};

END OBJECT {dimensions};

OBJECT AssetObj;
```

```
{----------------}
 ASK METHOD ObjInit;
{----------------}
BEGIN
NEW(assetDimens);
NEW(loadDimens);
NEW(cargoHold);
END METHOD {ObjInit};

{----------------}
 ASK METHOD ObjTerminate;
{----------------}
VAR
        cargo : CargoObj;
BEGIN
DISPOSE(assetDimens);
DISPOSE(loadDimens);
WHILE(ASK cargoHold numberIn > 0);
        cargo := ASK cargoHold TO Remove ();
        DISPOSE(cargo);
END WHILE;
DISPOSE(cargoHold);
IF(MMBF < > NILARRAY)
        DISPOSE(MMBF);
END IF;
IF(FailureVariateStream < > NILOBJ);
        DISPOSE(FailureVariateStream);
END IF;
END METHOD {ObjTerminate};

{----------------------------------------}
 ASK METHOD GetName (IN NewName : DescriptType;
                     IN i : INTEGER;
                     IN Unit : STRING);
{----------------------------------------}
BEGIN
        model := NewName;
        bumperNumber := ((ASK SELF model) + "_ "  + (Unit) + "_" + INTTOSTR(i));
        home := Unit;
END METHOD {GetName};

{----------------------------------------------------}
 ASK METHOD GetFields(IN AssetFieldsSHRec : SHierRecType);
{----------------------------------------------------}
VAR
        Height, Length, Width, CubeFt, Weight : REAL;
        AssetDimens : dimensionsObj;
        TravelRatesSHRec : SHierRecType;
        i, j : INTEGER;
        Rate : REAL;
        roadSurface, terrain : roadCharact;
BEGIN
NEW(AssetDimens);
IF (AssetFieldsSHRec = NILREC)
        ASK TraceStream TO WriteString("AssetFieldsSHRec is NILREC");
ELSE
        vehType := AssetFieldsSHRec.TopString;
        i := 1;
        model := (AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        type := (AssetFieldsSHRec.OwnedString[i]);
```

```
        INC(i);
        fuelCap := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        fuelConsump := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        odometer := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        engHrs := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        assetWeight := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        assetLength := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        Height := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        Length := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        Width := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        CubeFt := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        Weight := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        ASK AssetDimens TO GetFields(Height, Length, Width, CubeFt,  Weight);
        assetDimens := AssetDimens;
        maintManHours := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        adminLogTime := STRTOREAL(AssetFieldsSHRec.OwnedString[i]);
        INC(i);
        missionCapable := TRUE;
        NEW(FailureVariateStream);
        ASK FailureVariateStream TO SetSeed(ReadSeed());
        ASK SELF TO GetFailure(AssetFieldsSHRec);
{--------Getting Rates Of Travel for asset----------}

        FindSHRec(TravelRatesSHArray, model, TravelRatesSHRec);
        NEW(rateOfTravel, concrete..dirt, flat..mountainous);
        j := 1;
        FOR roadSurface := concrete TO dirt
                FOR terrain := flat TO mountainous
                        rateOfTravel[roadSurface, terrain] := STRTOREAL (TravelRatesSHRec.OwnedString[j]);
                INC(j);
                END FOR;
        END FOR;
END IF;
END METHOD {GetFields};

{-----------------------------------------------------------}
 ASK METHOD GetFailure(IN AssetFieldsSHRec : SHierRecType);
{-----------------------------------------------------------}
VAR
        i : INTEGER;
BEGIN
IF (AssetFieldsSHRec = NILREC)
        ASK TraceStream TO WriteString("AssetFieldsSHRec is NILREC");
ELSE
        NEW (MMBF, 1..1);
        MMBF[1] := STRTOREAL(AssetFieldsSHRec.OwnedString[16]);
        milesToFail := ASK FailureVariateStream Exponential (MMBF[1]);
END IF;
END METHOD {GetFailure};
```

```
{----------------------------}
 ASK METHOD CheckForBreakdown;
{----------------------------}
BEGIN
IF(odometer > = milesToFail)
        missionCapable := FALSE;
END IF;
END METHOD {CheckForBreakdown};

{---------------}
 ASK METHOD Break;
{---------------}
BEGIN
missionCapable := FALSE;
END METHOD {Break};

{-----------------------}
 ASK METHOD ResetOdometer;
{-----------------------}
BEGIN
odometer := 0.0;
tripOdometer := 0.0;
tempFix := FALSE;
milesToFail := ASK FailureVariateStream Exponential (MMBF[1]);
END METHOD {ResetOdometer};

{----------------------------}
 ASK METHOD ResetTripOdometer;
{----------------------------}
BEGIN
tripOdometer := 0.0;
END METHOD {ResetTripOdometer};

{-------------------------}
 ASK METHOD FixTemporarily;
{-------------------------}
BEGIN
tempFix := TRUE;
END METHOD {FixTemporarily};

{---------------------------------------}
 TELL METHOD LoadCargo (IN cargo : CargoObj);
{---------------------------------------}
BEGIN
ASK cargoHold TO Add (cargo);
END METHOD {LoadCargo};

{-----------------------------------------------}
 TELL METHOD UnloadCargo (IN weight : REAL;
                          IN length : REAL);
{-----------------------------------------------}
BEGIN
ASK loadDimens TO UpdateLoadWeight (-(weight));
ASK loadDimens TO UpdateLengthLoad(-(length));
END METHOD {UnloadCargo};

{-------------------------------------}
 ASK METHOD Refuel (OUT fuelUsed : REAL);
{-------------------------------------}
BEGIN
fuelUsed := fuelCap - fuelLevel;
```

```
fuelLevel := fuelLevel + fuelUsed;
END METHOD {Refuel};

{----------------------}
 ASK METHOD GetBigValue;
{----------------------}
VAR
AssetDimens : dimensionsObj;
BEGIN
NEW(AssetDimens);
ASK AssetDimens TO GetBigValue;
assetDimens := AssetDimens;
END METHOD {GetBigValue};

{------------------------}
 ASK METHOD GetSmallValue;
{------------------------}
VAR
AssetDimens : dimensionsObj;
BEGIN
NEW(AssetDimens);
ASK AssetDimens TO GetSmallValue;
assetDimens := AssetDimens;
END METHOD {GetSmallValue};

{---------------------------------------}
 ASK METHOD UpdateGuages (IN Link : LinkObj);
{---------------------------------------}
BEGIN
        odometer := odometer + ASK Link distance;
        tripOdometer := tripOdometer + ASK Link distance;
        fuelLevel := fuelLevel - (ASK Link distance/fuelConsump);
        {engHrs :=  engHrs + (ASK Link distance/rateOfTravel);}
        fuelGuage := fuelLevel/fuelCap;
END METHOD {UpdateGuages};

END OBJECT {asset};

OBJECT VehicleObj;
END OBJECT {VehicleObj};

OBJECT TruckObj;
END OBJECT {TruckObj};

OBJECT TracTrlObj;
END OBJECT {TracTrlObj};

OBJECT TracLowBoyObj;
END OBJECT {TracLowBoyObj};

OBJECT HettObj;
END OBJECT {HettObj};

OBJECT RecoveryObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
NEW(assetDimens);
NEW(loadDimens);
NEW(cargoHold);
```

```
NEW(RecoveryLoad);
NEW(FailureVariateStream);
ASK FailureVariateStream TO SetSeed(ReadSeed());
END METHOD {ObjInit};

{-------------------------------------------}
 TELL METHOD HookUp (IN brokenAsset : AssetObj);
{-------------------------------------------}
BEGIN
ASK RecoveryLoad TO Add (brokenAsset);
WAIT DURATION hookTime
END WAIT;
END METHOD {HookUp};

{-----------------}
 TELL METHOD UnHook;
{-----------------}
BEGIN
WAIT DURATION hookTime
END WAIT;
END METHOD {UnHook};

END OBJECT {RecoveryObj};

OBJECT VesselObj;
    END OBJECT {VesselObj};

OBJECT AirObj;
    END OBJECT {AirObj};

OBJECT TrainObj;
    END OBJECT {TrainObj};

END {IMPLEMENTATION} MODULE {asset}.

{*********************************************************}

DEFINITION MODULE CARGO;
{Methods and fields for cargo that is transported within ITTSS are defined here}
  FROM GLOBAL IMPORT HqNameType, ModelNameType, Dimensions, SupplyClassType,
          DescriptType, DestinationType, AssetTypeQueue, CargoTypeQueue,  NodeNameType, VehicleType;

TYPE
cargoList = CargoTypeQueue;
CargoObj = OBJECT
        priority      : REAL;
        height : REAL;
        width : REAL;
        length : REAL;
        cubeFt : REAL;
        weight : REAL;
        classOfSupply: SupplyClassType;
       destination  : STRING;
        origin : STRING;
        MOR : STRING {VehicleType};    {Method Of Resupply}
        nonseparable : BOOLEAN;

        ASK METHOD DumpStatus;
        ASK METHOD SeparateCargo (IN max : REAL;
                                  OUT separatedCargo : CargoObj);
        ASK METHOD ChangeWeight (IN max : REAL);
```

```
        ASK METHOD GetSupplyFields (IN ClassOfSupply : STRING;
                                    IN Weight : REAL;
                                    IN Length : REAL;
                                    IN MOR : STRING;
                                    IN Priority : REAL;
                                    IN SepStatus : BOOLEAN);
        ASK METHOD Adjust (IN Adjustment : REAL);

END OBJECT {cargo};

END {DEFINITION} MODULE {cargo}.

IMPLEMENTATION MODULE CARGO;
  FROM GLOBAL IMPORT HqNameType, NodeNameType, ModelNameType, Dimensions,
        SupplyClassType, DescriptType, DestinationType, AssetTypeQueue, CargoTypeQueue;
  FROM Debug IMPORT TraceStream;

OBJECT CargoObj;
{-------------------------------------------------------}
 ASK METHOD SeparateCargo (IN max : REAL;
                           OUT separatedCargo : CargoObj);
{-------------------------------------------------------}
VAR
        newWeight : REAL;
BEGIN
NEW(separatedCargo);
ASK separatedCargo TO GetSupplyFields(SELF.classOfSupply, max, 0.0,  SELF.MOR, SELF.priority, FALSE);
newWeight := (ASK SELF weight) - max;
ASK SELF TO ChangeWeight(newWeight);
END METHOD {SeparateCargo};

{-------------------------------------}
 ASK METHOD ChangeWeight (IN max : REAL);
{-------------------------------------}
BEGIN
        weight := max;
END METHOD {ChangeWeight};

{-------------------------------------------------}
ASK METHOD GetSupplyFields (IN class : SupplyClassType;
                            IN Weight : REAL;
                            IN Length : REAL;
                            IN methodOfResupply : STRING;
                            IN PRIORITY : REAL;
                            IN SepStatus : BOOLEAN);
{-------------------------------------------------}
BEGIN
        classOfSupply := class;
        weight := Weight;
        length := Length;
        MOR := methodOfResupply;
        priority := PRIORITY;
        nonseparable := SepStatus;
END METHOD {GetSupplyFields};

{---------------------------------------}
 ASK METHOD Adjust(IN Adjustment : REAL);
{---------------------------------------}
BEGIN
        weight := weight + Adjustment;
END METHOD;
```

```
END OBJECT {cargo};

END {IMPLEMENTATION} MODULE {cargo}.

{*********************************************************}

DEFINITION MODULE MOTORPL;
{Fields and methods of a motorpool are defined here.  This include matching cargo to assets, scheduling missions and conducting
them}
  FROM IOMod IMPORT StreamObj;
  FROM GrpMod IMPORT QueueObj;
  FROM GLOBAL IMPORT NodeNameType, AssetTypeQueue, CargoTypeQueue, RouteTypeQueue, NodeTypeQueue;
  FROM ASSET IMPORT AssetObj;
  FROM CARGO IMPORT CargoObj;
  FROM CONVOY IMPORT ConvoyObj;
  FROM RECORDS IMPORT VehiclesRec;
  FROM ROUTE IMPORT RouteObj;
  FROM NODE IMPORT NodeObj;

EXPORTTYPE
        MotorpoolObj = OBJECT; FORWARD;
TYPE
ConvoyTypeQueue = QueueObj;

RecoveryMissionObj = OBJECT
        location : STRING;
        numberToRecover : INTEGER;

        ASK METHOD GetFields (IN Location : STRING;
                              IN NumberToRecover : INTEGER);
        ASK METHOD Adjust (IN Adjustment : INTEGER);
END OBJECT {RecoveryMission};

MotorpoolObj = OBJECT
        name : NodeNameType;
        location : STRING;
        maxNumbInConvoy : INTEGER;
        distBetweenVeh : REAL;
        distBetweenConvoys : REAL;
        wreckersConvoy : BOOLEAN;
        repairTime : REAL;
        VehiclesRecord : VehiclesRec;
        AssetList : AssetTypeQueue;
        MasterRouteList : RouteTypeQueue;
        MasterNodeList : NodeTypeQueue;
        RecoveryMissionList : AssetTypeQueue;
        WaitingForAssetQueue : CargoTypeQueue;
        Dispatch : StreamObj;
        AlgorithmQueue : AssetTypeQueue;

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        ASK METHOD DumpStatus;
        ASK METHOD GetFields (IN name : STRING);
        ASK METHOD GetAssets (IN name : STRING);
  ASK METHOD GetMasterLists (INOUT MasterRouteList : RouteTypeQueue;
                             INOUT MasterNodeList : NodeTypeQueue);
        TELL METHOD ScheduleMission(IN Requestor : STRING;
                                    IN CargoToLoad : CargoTypeQueue;
                                    IN Convoy : ConvoyObj);
        TELL METHOD ConductMission (IN Convoy : ConvoyObj;
```

```
                                IN origin : STRING;
                                IN destination : STRING);
        ASK METHOD ReturnAssets (IN Convoy : ConvoyObj);
        TELL METHOD MatchCargoToAsset(IN asset : AssetObj;
                                IN load : CargoObj);
        TELL METHOD ConductLoading(IN Convoy : ConvoyObj);
        TELL METHOD SchedRecovery (IN Location : STRING;
                                IN NumberToRecover : INTEGER);
        TELL METHOD ConductRecovery (IN convoy : ConvoyObj;
                                IN origin : STRING;
                                IN destination : STRING);
        ASK METHOD FindWrecker (IN Convoy : ConvoyObj;
                                IN Mission : RecoveryMissionObj);
        ASK METHOD CheckWreckersReturning (IN Convoy : ConvoyObj;
                                INOUT NewConvoy : ConvoyObj);
        TELL METHOD ReturnFixedAsset (IN fixedAsset : AssetObj);
        TELL METHOD ReportStatus (IN MotorpoolReport : StreamObj);
        TELL METHOD ClearConvoy (IN Convoy : ConvoyObj;
                                IN Route : RouteObj);

END OBJECT {MotorpoolObj};

END {DEFINITION} MODULE {motorpl}.

IMPLEMENTATION MODULE MOTORPL;
  FROM UtilMod IMPORT Delay;
  FROM SimMod IMPORT SimTime;
  FROM IOMod IMPORT StreamObj, FileUseType(Output);
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT  NodeNameType, AssetTypeQueue, CargoTypeQueue, VehicleType,
        LinkTypeQueue, NodeTypeQueue, RouteTypeQueue;
  FROM RGLOBAL IMPORT AssetOwnersSHArray, MotorpoolSHArray, SHierRecType;
  FROM CREATEA IMPORT CreateAsset;
  FROM ASSET IMPORT AssetObj, VehicleObj, TruckObj, TracTrlObj, TracLowBoyObj, HettObj, OldHettObj, RecoveryObj;
  FROM FINDSH IMPORT FindSHRec;
  FROM CARGO IMPORT CargoObj;
  FROM FINDSHP IMPORT FindShortestPath;
  FROM CONVOY IMPORT ConvoyObj;
  FROM ROUTE IMPORT LinkObj, RouteObj;
  FROM CHECKAS IMPORT FindAssets, CheckReturningAssets, CheckLoadedAssets;
  FROM NODE IMPORT NodeObj, FindNod;
  FROM RECORDS IMPORT DispatchRecList, DispatchRec, VehTypeRecList, VehTypeRec, VehiclesRec;
  FROM FINDSUP IMPORT FindAssetTypeRecord;
  FROM SUPPLY IMPORT SupplyPointObj;


OBJECT RecoveryMissionObj;
{-------------------------------------------------}
 ASK METHOD GetFields (IN Location : STRING;
                IN NumberToRecover : INTEGER);
{-------------------------------------------------}
BEGIN
  location := Location;
        numberToRecover := NumberToRecover;
END METHOD {GetFields};

{-----------------------------------------}
 ASK METHOD Adjust (IN Adjustment : INTEGER);
{-----------------------------------------}
BEGIN
        numberToRecover := numberToRecover + Adjustment;
```

```
END METHOD {Adjust};

END OBJECT {RecoveryMission};

OBJECT MotorpoolObj;

{------------------}
 ASK METHOD ObjInit;
{------------------}
BEGIN
        NEW(AssetList);
        NEW(MasterRouteList);
        NEW(MasterNodeList);
        NEW(RecoveryMissionList);
        NEW(WaitingForAssetQueue);
        NEW(VehiclesRecord);
        NEW(AlgorithmQueue);
        NEW(Dispatch);
END METHOD {ObjInit};

{--------------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{--------------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{--------------------------------------}
 ASK METHOD GetFields (IN name : STRING);
{--------------------------------------}
VAR
        i : INTEGER;
        MotorpoolSHRec : SHierRecType;
        wreckersAccompany : STRING;
        smallestAsset, biggestAsset : AssetObj;
BEGIN
location := name;
FindSHRec(MotorpoolSHArray, name, MotorpoolSHRec);
i := 1;
REPEAT
        maxNumbInConvoy := STRTOINT (MotorpoolSHRec.OwnedString[i]);
        INC(i);
        distBetweenVeh := STRTOREAL (MotorpoolSHRec.OwnedString[i]);
        INC(i);
        distBetweenConvoys := STRTOREAL (MotorpoolSHRec.OwnedString[i]);
        INC(i);
        wreckersAccompany := (MotorpoolSHRec.OwnedString[i]);
        INC(i);
        IF(wreckersAccompany = "TRUE")
                wreckersConvoy := TRUE;
        END IF;
        repairTime := STRTOREAL(MotorpoolSHRec.OwnedString[i]);
        INC(i);
        NEW(biggestAsset);
        ASK biggestAsset TO GetName("BIG",0,location);
        ASK biggestAsset TO GetSmallValue;
        ASK AlgorithmQueue TO Add(biggestAsset);
        NEW(smallestAsset);
        ASK smallestAsset TO GetName("SMALL",0,location);
        ASK smallestAsset TO GetBigValue;
        ASK AlgorithmQueue TO Add(smallestAsset);
```

```
UNTIL ((i > HIGH(MotorpoolSHRec.OwnedString)) OR
        (MotorpoolSHRec.OwnedString[i] = "EOF"));
END METHOD {GetFields};

{-------------------------------------}
 ASK METHOD GetAssets (IN name : STRING);
{-------------------------------------}
VAR
        i,j,k : INTEGER;
        number, numberOfAssets : INTEGER;
        AssetOwnersSHRec : SHierRecType;
        model : STRING;
        truck : TruckObj;
        tracTrl : TracTrlObj;
        tracLowBoy : TracLowBoyObj;
        hett : HettObj;
        oldHett : OldHettObj;
        wrecker : RecoveryObj;
        vehicleTypeRecord : VehTypeRec;
BEGIN
FindSHRec(AssetOwnersSHArray, name, AssetOwnersSHRec);
i := 1;
WHILE ((AssetOwnersSHRec.OwnedString[i] < > "TRUCKS") AND
        (AssetOwnersSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
IF (AssetOwnersSHRec.OwnedString[i] = "none")
ELSE
  NEW(vehicleTypeRecord); {Getting record for all TRUCKS};
  REPEAT
                model := (AssetOwnersSHRec.OwnedString[i]);
                INC(i);
                number := STRTOINT(AssetOwnersSHRec.OwnedString[i]);
                FOR j := 1 TO number
                        NEW(truck);
                        CreateAsset ("TRUCKS", model, j, name, truck);
                        ASK AssetList TO Add (truck);
                        numberOfAssets := numberOfAssets + 1;
                END FOR;
                INC(i);
   UNTIL ((i > HIGH(AssetOwnersSHRec.OwnedString)) OR
                (AssetOwnersSHRec.OwnedString[i] = "\\") OR
                (AssetOwnersSHRec.OwnedString[i] = "TRAC_TRLS"));
   ASK vehicleTypeRecord TO GetFields("TRUCKS", numberOfAssets);
   ASK VehiclesRecord.vehTypeRecList TO Add(vehicleTypeRecord);
END IF;
numberOfAssets := 0;
WHILE ((AssetOwnersSHRec.OwnedString[i] < > "TRAC_TRLS") AND
        (AssetOwnersSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
IF (AssetOwnersSHRec.OwnedString[i] = "none")
ELSE
  NEW(vehicleTypeRecord); {Getting record for all TRAC_TRLS};
  REPEAT
                model := (AssetOwnersSHRec.OwnedString[i]);
                INC(i);
                number := STRTOINT(AssetOwnersSHRec.OwnedString[i]);
                FOR j := 1 TO number
```

```
                    NEW(tracTrl);
                    CreateAsset ("TRAC_TRLS", model, j, name, tracTrl);
                    ASK AssetList TO Add (tracTrl);
                    numberOfAssets := numberOfAssets + 1;
                END FOR;
                INC(i);
  UNTIL ((i > HIGH(AssetOwnersSHRec.OwnedString)) OR
                (AssetOwnersSHRec.OwnedString[i] = "\\") OR
                (AssetOwnersSHRec.OwnedString[i] = "TRAC_LOWBOYS"));
  ASK vehicleTypeRecord TO GetFields("TRAC_TRLS", numberOfAssets);
  ASK VehiclesRecord.vehTypeRecList TO Add(vehicleTypeRecord);
END IF;
numberOfAssets := 0;
WHILE ((AssetOwnersSHRec.OwnedString[i] < > "TRAC_LOWBOYS") AND
        (AssetOwnersSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
IF (AssetOwnersSHRec.OwnedString[i] = "none")
ELSE
 NEW(vehicleTypeRecord); {Getting record for all TRAC_LOWBOYS};
 REPEAT
                model := (AssetOwnersSHRec.OwnedString[i]);
                INC(i);
                number := STRTOINT(AssetOwnersSHRec.OwnedString[i]);
                FOR j := 1 TO number
                        NEW(tracLowBoy);
                        CreateAsset ("TRAC_LOWBOYS", model, j, name, tracLowBoy);
                        ASK AssetList TO Add (tracLowBoy);numberOfAssets := numberOfAssets + 1;
                END FOR;
                INC(i);
  UNTIL ((i > HIGH(AssetOwnersSHRec.OwnedString)) OR
                (AssetOwnersSHRec.OwnedString[i] = "\\") OR
                (AssetOwnersSHRec.OwnedString[i] = "HETTS"));
  ASK vehicleTypeRecord TO GetFields("TRAC_LOWBOYS", numberOfAssets);
  ASK VehiclesRecord.vehTypeRecList TO Add(vehicleTypeRecord);
END IF;
numberOfAssets := 0;
WHILE ((AssetOwnersSHRec.OwnedString[i] < > "HETTS") AND
        (AssetOwnersSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
IF (AssetOwnersSHRec.OwnedString[i] = "none")
ELSE
 NEW(vehicleTypeRecord); {Getting record for all HETTS};
 REPEAT
                model := (AssetOwnersSHRec.OwnedString[i]);
                INC(i);
                number := STRTOINT(AssetOwnersSHRec.OwnedString[i]);
                FOR j := 1 TO number
                        NEW(hett);
                        CreateAsset ("HETTS", model, j, name, hett);
                        ASK AssetList TO Add (hett);
                        numberOfAssets := numberOfAssets + 1;
                END FOR;
                INC(i);
  UNTIL ((i > HIGH(AssetOwnersSHRec.OwnedString)) OR
                (AssetOwnersSHRec.OwnedString[i] = "\\") OR
                (AssetOwnersSHRec.OwnedString[i] = "WRECKERS"));
  ASK vehicleTypeRecord TO GetFields("HETTS", numberOfAssets);
```

```
  ASK VehiclesRecord.vehTypeRecList TO Add(vehicleTypeRecord);
END IF;
numberOfAssets := 0;
WHILE ((AssetOwnersSHRec.OwnedString[i] < > "WRECKERS") AND
        (AssetOwnersSHRec.OwnedString[i] < > "\\"))
        INC(i);
END WHILE;
INC(i);
IF (AssetOwnersSHRec.OwnedString[i] = "none")
ELSE
 NEW(vehicleTypeRecord); {Getting record for all WRECKERS};
 REPEAT
                model := (AssetOwnersSHRec.OwnedString[i]);
                INC(i);
                number := STRTOINT(AssetOwnersSHRec.OwnedString[i]);
                FOR j := 1 TO number
                        NEW(wrecker);
                        CreateAsset ("WRECKERS", model, j, name, wrecker);
                        ASK AssetList TO Add (wrecker);
                        numberOfAssets := numberOfAssets + 1;
                END FOR;
                INC(i);
  UNTIL ((i > HIGH(AssetOwnersSHRec.OwnedString)) OR
                (AssetOwnersSHRec.OwnedString[i] = "\\") OR
                (AssetOwnersSHRec.OwnedString[i] = "TRUCKS"));
  ASK vehicleTypeRecord TO GetFields("WRECKERS",numberOfAssets);
  ASK VehiclesRecord.vehTypeRecList TO Add(vehicleTypeRecord);
END IF;
END METHOD {GetAssets};

{-------------------------------------------------------------}
 ASK METHOD GetMasterLists (INOUT masterRouteList : RouteTypeQueue;      INOUT masterNodeList :  NodeTypeQueue);
{-------------------------------------------------------------}
BEGIN
MasterRouteList := masterRouteList;
MasterNodeList := masterNodeList;
END METHOD {GetMasterLists};

{-------------------------------------------------------------}
 TELL METHOD ScheduleMission (IN Requestor : STRING;
                        IN CargoToLoad : CargoTypeQueue;
                        IN convoy : ConvoyObj);
{-------------------------------------------------------------}
VAR
        load : CargoObj;
        asset : AssetObj;
        newConvoy, returnConvoy : ConvoyObj;
        ConvoyList : ConvoyTypeQueue;
        RouteList : RouteObj;
        anotherTrip : BOOLEAN;
        CouldNotLoadQueue : AssetTypeQueue;
BEGIN
IF(convoy = NILOBJ)
        NEW(convoy);
        ASK convoy TO GetMissionType ("Resupply", location,  distBetweenConvoys, wreckersConvoy, repairTime);
END IF;
NEW(newConvoy);
ASK newConvoy TO GetMissionType ("Resupply", location, distBetweenConvoys, wreckersConvoy, repairTime);
NEW(CouldNotLoadQueue);
WHILE ASK CargoToLoad numberIn > 0
        load := ASK CargoToLoad TO Remove ();
```

```
        FindAssets(SELF, convoy, load, CouldNotLoadQueue);
END WHILE;
WHILE ASK CouldNotLoadQueue numberIn > 0
        load := ASK CouldNotLoadQueue TO Remove ();
        ASK WaitingForAssetQueue TO Add (load);
END WHILE;
DISPOSE(CouldNotLoadQueue);
NEW(returnConvoy);
asset := ASK convoy.AssetList First ();
REPEAT
        IF(asset.loadDimens.weight = 0.0)
                ASK convoy.AssetList TO RemoveThis (asset);
                ASK returnConvoy.AssetList TO Add (asset);
                IF((ASK convoy.AssetList numberIn) > 0)
                        asset := ASK convoy.AssetList First();
                ELSE
                        asset := NILOBJ;
                END IF;
        ELSE
                asset := ASK convoy.AssetList Next(asset);
        END IF;
UNTIL(asset = NILOBJ);
IF(ASK returnConvoy.AssetList numberIn > 0)
        ReturnAssets(returnConvoy);
ELSE
        DISPOSE(returnConvoy);
END IF;
IF(ASK convoy.AssetList numberIn > 0)
        WAIT FOR SELF TO ConductLoading(convoy);
        END WAIT;
        IF((ASK convoy.AssetList numberIn) > maxNumbInConvoy)
          REPEAT
                REPEAT
                        asset := ASK convoy.AssetList TO Remove(); ASK newConvoy.AssetList TO Add(asset);
                UNTIL((ASK convoy.AssetList numberIn) = maxNumbInConvoy);
                ASK convoy TO GetLength(distBetweenVeh);
                TELL SELF TO ConductMission(convoy, location, Requestor);
                convoy := newConvoy;
                NEW(newConvoy);
                ASK newConvoy TO GetMissionType ("Resupply", location,  distBetweenConvoys,wreckersConvoy,repairTime);
          UNTIL((ASK convoy.AssetList numberIn) <= maxNumbInConvoy);
          ASK convoy TO GetLength(distBetweenVeh);
          TELL SELF TO ConductMission(convoy, location, Requestor);
        ELSE
          ASK convoy TO GetLength(distBetweenVeh);
          TELL SELF TO ConductMission(convoy, location, Requestor);
        END IF;
ELSE
        DISPOSE(convoy);
END IF;
END METHOD {ScheduleMission};

{-----------------------------------------}
 ASK METHOD ReturnAssets (IN Convoy : ConvoyObj);
{-----------------------------------------}
VAR
        asset : AssetObj;
        VehTypeRecord : VehTypeRec;
BEGIN
asset := ASK Convoy.AssetList First();
IF (asset <> NILOBJ)
```

```
        REPEAT
                ASK Convoy.AssetList TO RemoveThis (asset);
                FindAssetTypeRecord(asset.vehType,SELF.VehiclesRecord,VehTypeRecord);
                ASK VehTypeRecord TO GetUncommitted;
                ASK AssetList TO Add (asset);
                IF ((ASK Convoy.AssetList numberIn) > 0)
                        asset := ASK Convoy.AssetList First ();
                ELSE
                        asset := NILOBJ;
                END IF;
        UNTIL (asset = NILOBJ);
END IF;
END METHOD {ReturnAssets};

{-------------------------------------------------}
 TELL METHOD ConductMission (IN convoy : ConvoyObj;
                        IN origin : STRING;
                        IN destination : STRING);
{-------------------------------------------------}
VAR
        asset : AssetObj;
        load : CargoObj;
        Link : LinkObj;
        Node : NodeObj;
        Route, ReturnRoute : RouteObj;
        routeDist, returnRouteDist, distance : REAL;
        maxRange, MaxRange : REAL;
        updatedOdometer, updatedFuelLevel : REAL;
        timeElapsed,
        mostTimeElapsed,
        TravelTime,
        clearanceDist : REAL;
BEGIN
FindShortestPath (MasterRouteList, origin, destination, Route);
FindShortestPath (MasterRouteList, destination, origin, ReturnRoute);
{Find total route distance}
Link := ASK Route.LinkRoute First ();
WHILE Link < > NILOBJ
        routeDist := routeDist + Link.distance;
        Link := ASK Route.LinkRoute Next (Link);
END WHILE;
Link := ASK ReturnRoute.LinkRoute First ();
WHILE Link < > NILOBJ
        returnRouteDist := returnRouteDist + Link.distance;
        Link := ASK ReturnRoute.LinkRoute Next (Link);
END WHILE;
distance := routeDist + returnRouteDist;
ASK convoy TO GetDistances(routeDist, returnRouteDist);
MaxRange := 9999999999999999.0;
WAIT FOR convoy TO Refuel (origin, MasterNodeList)
        asset := ASK convoy.AssetList First ();
        REPEAT
                maxRange := asset.fuelCap * asset.fuelConsump;
                IF(maxRange < MaxRange)
                        MaxRange := maxRange;
                END IF;
                asset := ASK convoy.AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        IF(MaxRange > distance)
                ASK convoy TO NotRefuelDuringMsn;
        END IF;
```

```
END WAIT;
FindNode(location, MasterNodeList, Node);
WAIT FOR Node.clearance TO Give (SELF, 1);
        WAIT FOR SELF TO ClearConvoy (convoy, Route);
        END WAIT;
        ASK Node.clearance TO TakeBack (SELF, 1);
END WAIT;
ASK convoy TO GetName(name + REALTOSTR(SimTime()));
ASK Dispatch TO WriteString("Convoy " + convoy.name + " is leaving " + name);
ASK Dispatch TO WriteLn;
ASK Dispatch TO WriteString("  TIME OF DEPARTURE is " + REALTOSTR (SimTime()));
ASK Dispatch TO WriteString(".  Number in convoy is " + INTTOSTR (ASK convoy.AssetList numberIn));
ASK Dispatch TO WriteLn;
ASK Dispatch TO WriteLn;
WAIT FOR  convoy TO Travel (destination, Route, MasterNodeList);
END WAIT;
{IF Fuelpoint avail, Refuel Assets}
IF (NOT convoy.fuelNotNeeded)
        WAIT FOR convoy TO Refuel (destination, MasterNodeList)
        END WAIT;
END IF;
FindNode(destination, MasterNodeList, Node);
WAIT FOR Node.clearance TO Give (SELF, 1);
        WAIT FOR SELF TO ClearConvoy (convoy, ReturnRoute);
        END WAIT;
        ASK Node.clearance TO TakeBack (SELF, 1);
END WAIT;

WAIT FOR convoy TO Travel (destination, ReturnRoute, MasterNodeList)
END WAIT;
ASK Dispatch TO WriteString("convoy " + SELF.name + "  returned to " + Node.name);
ASK Dispatch TO WriteLn;
ASK Dispatch TO WriteString("  time of RETURN is " + REALTOSTR (SimTime()));
ASK Dispatch TO WriteLn;
ASK Dispatch TO WriteString("  Number in convoy is " + INTTOSTR (ASK AssetList numberIn));
asset := ASK convoy.AssetList First ();

WAIT FOR convoy TO PerformPMCS (origin, MasterNodeList);
END WAIT;
IF(convoy.returnRouteDist > Node.maxMilesAllowed)
        WAIT DURATION Node.standDownTime
        END WAIT;
END IF;
IF ASK WaitingForAssetQueue numberIn > 0
        ScheduleMission(destination, WaitingForAssetQueue, convoy);
ELSE
        ReturnAssets(convoy);
END IF;
END METHOD {ConductMission};

{-----------------------------------------------}
 TELL METHOD MatchCargoToAsset(IN asset : AssetObj;
                               IN load : CargoObj);
{-----------------------------------------------}
BEGIN
WAIT FOR asset TO LoadCargo (load);
END WAIT;
END METHOD {MatchCargoToAsset};

{-----------------------------------------------}
 TELL METHOD ConductLoading(IN convoy : ConvoyObj);
```

```
{-----------------------------------------------}
VAR
        LoadingPt : SupplyPointObj;
        Node : NodeObj;
BEGIN
FindNode(location, MasterNodeList, Node);
WAIT FOR Node.mySupply.loadingPointsQueueTO PriorityGive (SELF, 1, convoy.priority);
        LoadingPt := ASK Node.mySupply.LoadingPointsTO Remove();
        WAIT FOR Node.mySupply TO LoadSupplies(convoy.AssetList, convoy.priority, LoadingPt);
        END WAIT;
END WAIT;
ASK Node.mySupply.loadingPointsQueueTO TakeBack (SELF, 1);
ASK Node.mySupply.LoadingPointsTO Add(LoadingPt);
END METHOD {ConductLoading};

{-----------------------------------------------------}
 TELL METHOD SchedRecovery (IN Destination : STRING;
                        IN NumberToRecover : INTEGER);
{-----------------------------------------------------}
VAR
        i : INTEGER;
        Node : NodeObj;
        brokenAsset : AssetObj;
        wrecker : RecoveryObj;
        convoy, newConvoy : ConvoyObj;
        RouteList : RouteObj;
        anotherTrip : BOOLEAN;
        RecoveryMission, NextMission : RecoveryMissionObj;
BEGIN
NEW(convoy);
ASK convoy TO GetMissionType ("Recovery", location, distBetweenConvoys, wreckersConvoy, repairTime);
NEW(newConvoy);
ASK newConvoy TO GetMissionType ("Recovery", location, distBetweenConvoys, wreckersConvoy, repairTime);
NEW(RecoveryMission);
ASK RecoveryMission TO GetFields (Destination, 0);
FOR i := 1 TO NumberToRecover
        FindWrecker(convoy, RecoveryMission);
END FOR;
IF((ASK convoy.AssetList numberIn) > 0)
        REPEAT
        WAIT FOR SELF TO ConductRecovery(convoy, location, Destination);
        END WAIT;
                anotherTrip := FALSE;
                FindNode(location, MasterNodeList, Node);
                wrecker := ASK convoy.AssetList First ();
                REPEAT
                        IF((ASK wrecker.RecoveryLoad numberIn) > 0);
                            WAIT FOR wrecker TO UnHook;
                                brokenAsset := ASK wrecker.RecoveryLoad TO Remove ();
                            END WAIT;
                            {got to unload cargo}
                            TELL Node.myMaintenance TO ReceiveWork (brokenAsset);
                        END IF;
                        wrecker := ASK convoy.AssetList Next (wrecker);
                UNTIL (wrecker = NILOBJ);
                IF(convoy.returnRouteDist > Node.maxMilesAllowed)
                        WAIT DURATION Node.standDownTime
                        END WAIT;
                END IF;
                ReturnAssets(convoy);
                IF((ASK newConvoy.AssetList numberIn) > 0)
```

```
                    convoy := newConvoy;
                    anotherTrip := TRUE;
                END IF;
        UNTIL (NOT anotherTrip);
END IF;
END METHOD {ScheduleRecovery};

{-----------------------------------------------------}
 TELL METHOD ConductRecovery (IN convoy : ConvoyObj;
                    IN origin : STRING;
                    IN destination : STRING);
{-----------------------------------------------------}
VAR
        asset : AssetObj;
        newConvoy : ConvoyObj;
        load : CargoObj;
        Link : LinkObj;
        Route,ReturnRoute : RouteObj;
        routeDist, returnRouteDist, distance : REAL;
        maxRange, MaxRange : REAL;
        updatedOdometer, updatedFuelLevel : REAL;
        timeElapsed,
        mostTimeElapsed : REAL;
        TravelTime : REAL;
        anotherTrip : BOOLEAN;
BEGIN
        FindShortestPath (MasterRouteList, origin, destination, Route);
        FindShortestPath (MasterRouteList, destination, origin,  ReturnRoute);
        {Find total route distance}
        Link := ASK Route.LinkRoute First ();
        WHILE Link < > NILOBJ
                routeDist := routeDist + Link.distance;
                Link := ASK Route.LinkRoute Next (Link);
        END WHILE;
        Link := ASK ReturnRoute.LinkRoute First ();
        WHILE Link < > NILOBJ
                returnRouteDist := returnRouteDist + Link.distance;
                Link := ASK ReturnRoute.LinkRoute Next (Link);
        END WHILE;
        distance := routeDist + returnRouteDist;
        ASK convoy TO GetDistances(routeDist, returnRouteDist);
        WAIT FOR convoy TO Refuel (origin, MasterNodeList)
                asset := ASK AssetList First ();
                REPEAT
                        maxRange := asset.fuelCap * asset.fuelConsump;
                        IF(maxRange < MaxRange)
                                MaxRange := maxRange;
                        END IF;
                        asset := ASK AssetList Next (asset);
                UNTIL (asset = NILOBJ);
                IF(MaxRange > distance)
                        ASK convoy TO NotRefuelDuringMsn;
                END IF;
        END WAIT;
        WAIT FOR  convoy TO Travel (destination, Route, MasterNodeList);
        END WAIT;
{IF Fuelpoint avail, Refuel Assets}
        IF (NOT convoy.fuelNotNeeded)
                WAIT FOR convoy TO Refuel (destination, MasterNodeList)
                END WAIT;
        END IF;
```

```
        WAIT FOR convoy TO Travel (destination, ReturnRoute, MasterNodeList)
        END WAIT;
        WAIT FOR convoy TO PerformPMCS (origin, MasterNodeList);
        END WAIT;
END METHOD {ConductRecovery};

{-------------------------------------------------------------}
 ASK METHOD FindWrecker (IN Convoy : ConvoyObj;
                         IN RecoveryMission : RecoveryMissionObj);
{-------------------------------------------------------------}
VAR
        wrecker : AssetObj;
        foundWrecker : BOOLEAN;
        VehTypeRecord : VehTypeRec;
BEGIN
foundWrecker := FALSE;
wrecker := ASK AssetList First ();
IF(wrecker < > NILOBJ)
        REPEAT
                IF(wrecker.type = "RECOVERY")
                        ASK AssetList TO RemoveThis (wrecker);
                        FindAssetTypeRecord(wrecker.vehType,SELF.VehiclesRecord, VehTypeRecord);
                        ASK Convoy.AssetList TO Add (wrecker);
                        foundWrecker := TRUE;
                ELSE
                        wrecker := ASK AssetList Next (wrecker);
                END IF;
        UNTIL ((wrecker = NILOBJ) OR (foundWrecker));
        IF(NOT foundWrecker)
                ASK RecoveryMission TO Adjust (1);
                ASK RecoveryMissionList TO Add (RecoveryMission);
        END IF;
END IF;
END METHOD {FindWrecker};

{-------------------------------------------------------------}
 ASK METHOD CheckWreckersReturning (IN Convoy : ConvoyObj;
                                    INOUT NewConvoy : ConvoyObj);
{-------------------------------------------------------------}
VAR
        asset : AssetObj;
        recoveryMission : RecoveryMissionObj;
BEGIN
asset := ASK Convoy.AssetList First ();
REPEAT
        recoveryMission := ASK RecoveryMissionList First ();
        REPEAT
                IF(recoveryMission.numberToRecover > 0)
                        REPEAT
                        ASK Convoy.AssetList TO RemoveThis (asset);
                        ASK NewConvoy.AssetList TO Add (asset);
                        ASK recoveryMission TO Adjust (-1);
                        IF ((ASK Convoy.AssetList numberIn) = 0)
                                asset := NILOBJ;
                        ELSE
                                asset := ASK Convoy.AssetList Next (asset);
                        END IF;
                ELSIF(asset < > NILOBJ)
                        recoveryMission := ASK RecoveryMissionList Next (recoveryMission);
                END IF;
        UNTIL ((recoveryMission = NILOBJ) OR (asset = NILOBJ);
```

```
        IF(asset < > NILOBJ)
                asset := ASK Convoy.AssetList Next (asset);
        END IF;
UNTIL (asset = NILOBJ);
END METHOD {CheckReturningWreckers};

{-------------------------------------------------------}
 TELL METHOD ReturnFixedAsset (IN fixedAsset : AssetObj);
{-------------------------------------------------------}
VAR
        convoy : ConvoyObj;
        Route : RouteObj;
        VehTypeRecord : VehTypeRec;
        Node : NodeObj;
        load : CargoObj;
BEGIN
NEW(convoy);
IF(fixedAsset.home = SELF.location)
        FindAssetTypeRecord(fixedAsset.vehType,SELF.VehiclesRecord, VehTypeRecord);
        ASK VehTypeRecord TO GetRepaired;
        ASK AssetList TO Add(fixedAsset);
ELSE
        FindNode(fixedAsset.home, MasterNodeList, Node);
        FindAssetTypeRecord(fixedAsset.type, Node.myMotorpool.VehiclesRecord, VehTypeRecord);
        ASK VehTypeRecord TO GetRepaired;
        ASK convoy TO GetMissionType("Repair", location, distBetweenConvoys, NOT wreckersConvoy, repairTime);
        ASK convoy.AssetList TO Add(fixedAsset);
        FindShortestPath (MasterRouteList, location, fixedAsset.home, Route);
        WAIT FOR convoy TO Travel (fixedAsset.home, Route, MasterNodeList);
        END WAIT;
END IF;
END METHOD {ReturnFixedAsset};

{-------------------------------------------------------}
 TELL METHOD ReportStatus (IN MotorpoolReport : StreamObj);
{-------------------------------------------------------}
VAR
        vehicleRec : VehTypeRec;
BEGIN
ASK MotorpoolReport TO WriteString("*************************************************************");
ASK MotorpoolReport TO WriteLn;
ASK MotorpoolReport TO WriteString("*******Status for " + name + " MOTORPOOL.  Time is " + REALTOSTR(SimTime()));
ASK MotorpoolReport TO WriteLn;
vehicleRec := ASK VehiclesRecord.vehTypeRecList First ();
REPEAT
        ASK MotorpoolReport TO WriteString("----------Vehicle Type is  " + vehicleRec.vehType);
        ASK MotorpoolReport TO WriteLn;
        ASK MotorpoolReport TO          WriteString("-----------------------------------------------------------------------------------");
        ASK MotorpoolReport TO WriteLn;
        ASK MotorpoolReport TO WriteString("Total Assets are     " +  INTTOSTR (vehicleRec.totalAssets));
        ASK MotorpoolReport TO WriteLn;
        ASK MotorpoolReport TO WriteString("Total Committed is " +  INTTOSTR (vehicleRec.totalCommitted));
        ASK vehicleRec TO FindRates;
        ASK MotorpoolReport TO WriteString("  Util Rate % is          " + REALTOSTR (vehicleRec.utilRate));
        ASK MotorpoolReport TO WriteLn;
        ASK MotorpoolReport TO WriteString("Total Available is    " +  INTTOSTR (vehicleRec.totalAvail));
        ASK MotorpoolReport TO WriteString("  Avail Rate % is        "  + REALTOSTR (vehicleRec.availRate));
        ASK MotorpoolReport TO WriteLn;
        ASK MotorpoolReport TO WriteString("Total Deadlined is  " +  INTTOSTR ( v e h i c l e R e c . t o t a l A s s e t s  -
vehicleRec.totalAvail));
        ASK MotorpoolReport TO WriteLn;
```

```
        ASK MotorpoolReport TO          WriteString("**************************************************");
        ASK MotorpoolReport TO WriteLn;
        vehicleRec := ASK VehiclesRecord.vehTypeRecList Next (vehicleRec);
UNTIL (vehicleRec = NILOBJ);
END METHOD {ReportStatus};
{-----------------------------------------}
 TELL METHOD ClearConvoy (IN Convoy : ConvoyObj;
                          IN Route : RouteObj);
{-----------------------------------------}
VAR
        asset : AssetObj;
        Link : LinkObj;
        timeElapsed,
        mostTimeElapsed : REAL;
        TravelTime : REAL;
        clearanceDist : REAL;
        clearanceTime : REAL;
        rate : REAL;
BEGIN
Link := ASK Route.LinkRoute First ();
asset := ASK AssetList First ();
TravelTime := 0.0;
REPEAT
        timeElapsed := ASK Link distance/ASK asset rateOfTravel [Link.roadSurface, Link.terrain];
        IF (timeElapsed > mostTimeElapsed)
                TravelTime := timeElapsed;
                rate := ASK asset rateOfTravel[Link.roadSurface, Link.terrain];
        END IF;
        asset := ASK AssetList Next (asset);
UNTIL (asset = NILOBJ);
clearanceDist := (Convoy.totalLength + Convoy.distBetweenConvoys)/5280.0;
clearanceTime := clearanceDist/rate;
WAIT DURATION clearanceTime
END WAIT;
END METHOD {ClearConvoy};

END OBJECT {MotorpoolObj};

END {IMPLEMENTATION} MODULE {motorpl}.

{*****************************************************}

DEFINITION MODULE RECORDS;
{Records that are kept for each vehicle type are defined here}
  FROM GrpMod IMPORT QueueObj;
  FROM GLOBAL IMPORT NodeNameType;

TYPE
{If individual records are required for each asset, use following...
VehRecList = QueueObj;
VehRec = OBJECT {Individual vehicle Record}
        bumperNumber : STRING;
        miles : REAL;
        totalMiles : REAL;
        opHours : REAL;
        totalOpHours : REAL;
        committed : BOOLEAN;
        deadlined : BOOLEAN;

        ASK METHOD GetFields(IN BumperNumber : STRING);
        ASK METHOD UpDate(IN MilesDriven : REAL;
```

```
                    IN OpHours : REAL);
        ASK METHOD GetCommitted;
        ASK METHOD ChangeDeadlinedStatus(IN Status : BOOLEAN);
        ASK METHOD ResetFields;
END OBJECT {VehRec};}

VehTypeRec = OBJECT {All vehicles of a particular TYPE}
        vehType : STRING;
        miles : REAL;
        totalMiles : REAL;
        opHours : REAL;
        totalOpHours : REAL;
        totalAssets : INTEGER;
        totalCommitted : INTEGER;
        totalAvail : INTEGER;
        shortTons : REAL;
        totalShortTons : REAL;
        numberOfPieces : INTEGER;
        availRate : REAL;
        utilRate : REAL;
        {vehRecList : VehRecList;  only needed if ind asset rec needed}

        ASK METHOD ObjInit;
        ASK METHOD GetFields(IN vehType : STRING;
                             IN totalAssets : INTEGER);
        ASK METHOD GetCommitted;
        ASK METHOD GetUncommitted;
        ASK METHOD GetDeadlined;
        ASK METHOD GetRepaired;
        ASK METHOD UpDate (IN Miles : REAL;
                           IN Hours : REAL;
                           IN STons : REAL);
        ASK METHOD Reset;
        ASK METHOD FindRates;

END OBJECT {VehTypeRec};

VehTypeRecList = QueueObj;

VehiclesRec = OBJECT(VehTypeRec); {All vehicles in transportation sys}
        vehTypeRecList : VehTypeRecList;
OVERRIDE
        ASK METHOD ObjInit;
END OBJECT{VehiclesRec};

END {DEFINITION} MODULE {records}.

IMPLEMENTATION MODULE RECORDS;
  FROM GLOBAL IMPORT NodeNameType;
  FROM Debug IMPORT TraceStream;
{For future development, if individual records are required...
OBJECT VehRec; {For each individual vehicle, M35A2, M55A1...}
{-----------------------------------------}
 ASK METHOD GetFields(IN BumperNumber : STRING);
{-----------------------------------------}
BEGIN
        bumperNumber := BumperNumber;
END METHOD {GetFields};

{-----------------------------------------}
 ASK METHOD UpDate(IN MilesDriven : REAL;
```

```
                IN OpHours : REAL);
{-------------------------------------}
BEGIN
        miles := miles + MilesDriven;
        totalMiles := totalMiles + MilesDriven;
        opHours := opHours + OpHours;
        totalOpHours := totalOpHours + OpHours;
END METHOD {UpDate};

{--------------------}
 ASK METHOD GetCommitted;
{--------------------}
BEGIN
        committed := TRUE;
END METHOD {GetCommitted};

{-------------------------------------------------}
 ASK METHOD ChangeDeadlinedStatus(IN Status : BOOLEAN);
{-------------------------------------------------}
BEGIN
        deadlined := Status;
END METHOD {ChangeDeadlinedStatus};

{--------------------}
 ASK METHOD ResetFields;
{--------------------}
BEGIN
        committed := FALSE;
        miles := 0.0;
        opHours := 0.0;
END METHOD {ResetFields};

END OBJECT {VehRec};}

OBJECT VehTypeRec; {For each type of vehicle, TRUCKS, TRACTORS...}
{----------------}
 ASK METHOD ObjInit;
{----------------}
BEGIN
        {NEW(vehRecList);     use if you do have a record list for each}
END METHOD {ObjInit};

{-----------------------------------------}
 ASK METHOD GetFields(IN VehType : STRING;
                IN TotalAssets : INTEGER);
{-----------------------------------------}
BEGIN
        vehType := VehType;
        totalAssets := TotalAssets;
        totalAvail := TotalAssets;
END METHOD {GetFields};

{--------------------}
 ASK METHOD GetCommitted;
{--------------------}
BEGIN
        totalCommitted := totalCommitted + 1;
END METHOD {GetCommitted};

{------------------------}
 ASK METHOD GetUncommitted;
```

```
{------------------------}
BEGIN
        totalCommitted := totalCommitted - 1;
END METHOD {GetUncommitted};

{----------------------}
 ASK METHOD GetDeadlined;
{----------------------}
BEGIN
        totalAvail := totalAvail - 1;
END METHOD {GetDeadlined};

{---------------------}
 ASK METHOD GetRepaired;
{---------------------}
BEGIN
        totalAvail := totalAvail + 1;
END METHOD {GetRepaired};

{-----------------------------------}
 ASK METHOD UpDate (IN Miles : REAL;
                    IN Hours : REAL;
                    IN STons : REAL);
{-----------------------------------}
BEGIN
        miles := miles + Miles;
        totalMiles := totalMiles + Miles;
        opHours := opHours + Hours;
        totalOpHours := totalOpHours + Hours;
        shortTons := shortTons + STons;
        totalShortTons := totalShortTons + STons;
availRate := FLOAT(totalAvail)/FLOAT(totalAssets);
utilRate := FLOAT(totalCommitted)/FLOAT(totalAssets);
END METHOD {UpDate};

{---------------}
 ASK METHOD Reset;
{---------------}
BEGIN
        miles := 0.0;
        opHours := 0.0;
        shortTons := 0.0;
        totalCommitted := 0;
        totalAvail := totalAssets;
END METHOD {Reset};

{-------------------}
 ASK METHOD FindRates;
{-------------------}
BEGIN
availRate := FLOAT(totalAvail)/FLOAT(totalAssets);
utilRate := FLOAT(totalCommitted)/FLOAT(totalAssets);
END METHOD {FindRates};

END OBJECT {VehTypeRec};

OBJECT VehiclesRec; {All major types of vehicles, SUM of everything}

{------------------}
 ASK METHOD ObjInit;
{------------------}
```

```
BEGIN
        NEW(vehTypeRecList);
END METHOD {ObjInit};

END OBJECT{VehiclesRec};

END {IMPLEMENTATION} MODULE {records}.

{*********************************************************}

DEFINITION MODULE MAINT;
{All fields and methods of a maintenance facility are defined here}
  FROM IOMod IMPORT StreamObj;
  FROM GLOBAL IMPORT NodeNameType, AssetTypeQueue, NodeTypeQueue, RouteTypeQueue;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM ASSET IMPORT AssetObj;

EXPORTTYPE
        MaintenanceObj = OBJECT; FORWARD;
TYPE
MaintenanceObj = OBJECT
        name : NodeNameType;
        WorkList : AssetTypeQueue;
        myMotorpool : MotorpoolObj;
        MaintReport : StreamObj;

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        ASK METHOD GetMyMotorpool (IN Motorpool : MotorpoolObj);
        TELL METHOD RecoverAssets (IN Location : STRING;
                                   IN NumberToRecover : INTEGER);
        TELL METHOD ReceiveWork (IN brokenAsset : AssetObj);
        TELL METHOD Repair (IN brokenAsset : AssetObj);

END OBJECT {Maintenance};

END {DEFINITION} MODULE {maintenance}.

IMPLEMENTATION MODULE MAINT;
  FROM IOMod IMPORT StreamObj, FileUseType(Output);
  FROM SimMod IMPORT SimTime;
  FROM GLOBAL IMPORT NodeNameType, AssetTypeQueue, RouteTypeQueue, NodeTypeQueue;
  FROM RGLOBAL IMPORT AssetOwnersSHArray, SHierRecType;
  FROM FINDSH IMPORT FindSHRec;
  FROM Debug IMPORT TraceStream;
  FROM ASSET IMPORT AssetObj;
  FROM CREATEA IMPORT CreateAsset;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM CONVOY IMPORT ConvoyObj;
  FROM ROUTE IMPORT RouteObj;

OBJECT MaintenanceObj;
{-------------------}
 ASK METHOD ObjInit;
{-------------------}
BEGIN
        NEW(WorkList);
        NEW(MaintReport);
        ASK MaintReport TO Open ("Maint.rpt", Output);
END METHOD {ObjInit};
```

```
{---------------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{---------------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{-----------------------------------------------------}
 ASK METHOD GetMyMotorpool (IN Motorpool :MotorpoolObj);
{-----------------------------------------------------}
BEGIN
        myMotorpool := Motorpool;
END METHOD {GetMyMotorpool};

{-----------------------------------------------------}
 TELL METHOD RecoverAssets (IN Location : STRING;
                        IN NumberToRecover : INTEGER);
{-----------------------------------------------------}
VAR
        brokenAsset : AssetObj;
        convoy, newConvoy : ConvoyObj;
        RouteList : RouteObj;
        anotherTrip : BOOLEAN;
BEGIN
TELL myMotorpool TO SchedRecovery (Location, NumberToRecover);
END METHOD {RecoverAssets};

{-------------------------------------------------}
 TELL METHOD ReceiveWork (IN brokenAsset : AssetObj);
{-------------------------------------------------}
BEGIN
ASK MaintReport TO WriteString("Maint " + name + " received " + brokenAsset.bumperNumber + " at " +
REALTOSTR(SimTime()));
ASK MaintReport TO WriteLn;
ASK MaintReport TO WriteLn;
WAIT FOR SELF TO Repair(brokenAsset)
END WAIT;
ASK MaintReport TO WriteString("MAINT " + name + " RELEASED " + brokenAsset.bumperNumber + " at " +
REALTOSTR(SimTime()));
ASK MaintReport TO WriteLn;
ASK MaintReport TO WriteLn;
END METHOD {ReceiveWork};

{-----------------------------------------}
 TELL METHOD Repair (IN brokenAsset : AssetObj);
{-----------------------------------------}
VAR
        repairTime : REAL;
BEGIN
repairTime := brokenAsset.adminLogTime + (brokenAsset.maintManHours * brokenAsset.odometer);
WAIT DURATION repairTime
END WAIT;
ASK brokenAsset TO ResetOdometer;
TELL myMotorpool TO ReturnFixedAsset(brokenAsset);
{send asset back}
END METHOD {Repair};

END OBJECT {Maintenance};

END {IMPLEMENTATION} MODULE {maint}.
```

```
{*********************************************************}

DEFINITION MODULE SUPPLY;
{Methods and fields for supply points are defined here, to include checking stock, resupplying, loading, unloading and receiving
supplies}
  FROM IOMod IMPORT StreamObj;
  FROM GrpMod IMPORT QueueObj;
  FROM ResMod IMPORT ResourceObj;
  FROM GLOBAL IMPORT AssetTypeQueue, NodeNameType, CargoTypeQueue, SupplyClassType,
        SupplyRecordTypeQueue, RequestTypeQueue, ConsumerQueue;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM REQUEST IMPORT RequestObj;
  FROM MOTORPL IMPORT  MotorpoolObj;
  FROM CARGO IMPORT CargoObj;

EXPORTTYPE
        SupplyObj = OBJECT; FORWARD;
TYPE

SupplyPointQueue = ResourceObj;
MaterialHandlingEquipmentQueue = ResourceObj;
ConvoyTypeQueue = QueueObj;
SupplyPointsQueue = QueueObj;

SupplyPointObj = OBJECT
        MHE : MaterialHandlingEquipmentQueue;
        ASK METHOD ObjInit;
END OBJECT {SupplyPointObj};

SupplyObj = OBJECT
        name : NodeNameType;
        location : STRING;
        myMotorpool : MotorpoolObj;
        mySupplySource : SupplyObj;
        inventory : CargoTypeQueue;
        supplyRecords : SupplyRecordTypeQueue;
        waiting : CargoTypeQueue;
        loadTime : REAL;
        unloadTime : REAL;
        cargoToLoad : CargoTypeQueue;
        timeToCheckStock : REAL;
        numberOfReceivers : INTEGER;
        numberOfLoaders : INTEGER;
        numberOfMHE : INTEGER;
        receivingPointsQueue : SupplyPointQueue;
        loadingPointsQueue : SupplyPointQueue;
        ReceivingPoints : SupplyPointsQueue;
        LoadingPoints : SupplyPointsQueue;
        convoyQueue : ConvoyTypeQueue;

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        ASK METHOD AdjustInventory (IN supplyItem : CargoObj;
                                    IN amount : REAL);
        ASK METHOD GetMyMotorpool(IN Motorpool : MotorpoolObj);
        ASK METHOD GetFields (IN Name : STRING);
        ASK METHOD GetConsumptionRates(IN Consumers : ConsumerQueue);
        ASK METHOD GetMotorpool(IN Motorpool : MotorpoolObj);
        ASK METHOD GetSupplyRecord(IN itemName : STRING;
                                   IN SupplyClass : STRING;
```

```
                        OUT Record : SupplyRecordObj;
                        IN nonseparable : BOOLEAN);
    ASK METHOD GetSupplySource (IN SupplySource : SupplyObj);
    TELL METHOD PrepareToStartSupply(IN dayToEndConsuming : INTEGER);
    TELL METHOD Resupply (IN RequestList : RequestTypeQueue);
    TELL METHOD CollectItems (IN RequestList : RequestTypeQueue);
    TELL METHOD CheckStock;
    TELL METHOD ReceiveSupplies (IN AssetList : AssetTypeQueue;
                        IN priority : REAL;
                        IN ReceivingPt : SupplyPointObj);
    TELL METHOD LoadSupplies (IN AssetList : AssetTypeQueue;
                        IN priority : REAL;
                        IN LoadingPoint : SupplyPointObj);

    ASK METHOD AdmitItem (IN SupplyRecord : SupplyRecordObj;
                        IN cargo : CargoObj);
    TELL METHOD ReportStatus (IN SupplyReport : StreamObj);

END OBJECT {Supply};

EconomyObj = OBJECT(SupplyObj);
OVERRIDE
    TELL METHOD Resupply (IN RequestList : RequestTypeQueue);
END OBJECT {Economy};

END {DEFINITION} MODULE {supply}.

IMPLEMENTATION MODULE SUPPLY;
  FROM SimMod IMPORT SimTime;
  FROM IOMod IMPORT StreamObj;
  FROM GLOBAL IMPORT NodeNameType, CargoTypeQueue, AssetTypeQueue, SupplyClassType,
       SupplyRecordTypeQueue, RequestTypeQueue, ConsumerQueue, DisposeOfQueue;
  FROM RGLOBAL IMPORT SHierRecType,SupplySHArray;
  FROM FINDSH IMPORT FindSHRec;
  FROM ASSET IMPORT AssetObj;
  FROM CARGO IMPORT CargoObj;
  FROM FINDSUP IMPORT FindCargo, FindItem, FillRequest, FindRec;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM DISPREQ IMPORT DisposeRequestTypeQueue;
  FROM REQUEST IMPORT RequestObj;
  FROM Debug IMPORT TraceStream;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM UNITS IMPORT ConsumerObj;

OBJECT SupplyPointObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
    NEW(MHE);
END METHOD {ObjInit};

END OBJECT {SupplyPointObj};

OBJECT SupplyObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
    NEW(inventory);
    NEW(supplyRecords);
```

```
        NEW(waiting);
        NEW(receivingPointsQueue);
        NEW(loadingPointsQueue);
        NEW(ReceivingPoints);
        NEW(LoadingPoints);
        NEW(convoyQueue);
END METHOD {ObjInit};

{---------------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{---------------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{-----------------------------------------------------}
 ASK METHOD AdjustInventory (IN supplyItem : CargoObj;
                        IN amount : REAL);
{-----------------------------------------------------}
BEGIN
IF(supplyItem.nonseparable)
        ASK inventory TO Add (supplyItem);
ELSE
        ASK supplyItem TO ChangeWeight(amount);
END IF;
END METHOD {AdjustInventory};

{-----------------------------------------------------}
 ASK METHOD GetMyMotorpool (IN Motorpool :MotorpoolObj);
{-----------------------------------------------------}
BEGIN
        myMotorpool := Motorpool;
END METHOD {GetMyMotorpool};

{-----------------------------------}
 ASK METHOD GetFields(IN Name : STRING);
{-----------------------------------}
VAR
        SupplyRecord : SupplyRecordObj;
        supplyClass : SupplyClassType;
        methodOfResupply : STRING;
        i, j, daysOSupply  : INTEGER;
        stocklvl, upperlvl, priority : REAL;
        cargo : CargoObj;
        SupplySHRec : SHierRecType;
        ReceivingPoint : SupplyPointObj;
        LoadingPoint : SupplyPointObj;
BEGIN
FindSHRec(SupplySHArray, Name, SupplySHRec);
        location := Name;
        j := 1;
        timeToCheckStock := STRTOREAL(SupplySHRec.OwnedString[j]);
        INC(j);
        numberOfReceivers := STRTOINT(SupplySHRec.OwnedString[j]);
        INC(j);
        numberOfLoaders := STRTOINT(SupplySHRec.OwnedString[j]);
        INC(j);
        numberOfMHE := STRTOINT(SupplySHRec.OwnedString[j]);
        INC(j);
        loadTime := STRTOREAL(SupplySHRec.OwnedString[j]);
        INC(j);
```

```
        unloadTime :=    STRTOREAL(SupplySHRec.OwnedString[j]);
        INC(j);

        ASK receivingPointsQueue TO Create(numberOfReceivers);

        FOR i := 1 TO numberOfReceivers
                NEW(ReceivingPoint);
                ASK ReceivingPoint.MHE TO Create (numberOfMHE);
                ASK ReceivingPoints TO Add (ReceivingPoint);
        END FOR;

        ASK loadingPointsQueue TO Create(numberOfLoaders);

        FOR i := 1 TO numberOfLoaders
                NEW(LoadingPoint);
                ASK LoadingPoint.MHE TO Create (numberOfMHE);
                ASK LoadingPoints TO Add (LoadingPoint);
        END FOR;

        REPEAT
                NEW(SupplyRecord);
                NEW(cargo);
                supplyClass := SupplySHRec.OwnedString[j];
                INC(j);
                methodOfResupply := SupplySHRec.OwnedString[j];INC(j);
                priority := STRTOREAL(SupplySHRec.OwnedString[j]);
                INC(j);
                stocklvl := STRTOREAL(SupplySHRec.OwnedString[j]);
                INC(j);
                daysOSupply := STRTOINT(SupplySHRec.OwnedString[j]);
                INC(j);
                upperlvl := STRTOREAL(SupplySHRec.OwnedString[j]);
                INC(j);
                ASK SupplyRecord TO GetFields (supplyClass, stocklvl, daysOSupply, upperlvl, FALSE);
                ASK supplyRecords TO Add (SupplyRecord);
                ASK cargo TO GetSupplyFields (supplyClass, stocklvl, 0.0, methodOfResupply, priority , FALSE);
                ASK inventory TO Add (cargo);
        UNTIL(SupplySHRec.OwnedString[j] = "EOF");

END METHOD {GetFields};

{-------------------------------------------------------}
 ASK METHOD GetConsumptionRates(IN Consumers : ConsumerQueue);
{-------------------------------------------------------}
VAR
        consumer : ConsumerObj;
        OutRec : SupplyRecordObj;
BEGIN
consumer := ASK Consumers First();
REPEAT
        FindRec(consumer.name, supplyRecords, OutRec);
        IF(OutRec = NILOBJ)
ASK TraceStream TO WriteString("**********Mistake, should have been a record matching consumer with
SupplyRecord ");
ASK TraceStream TO WriteLn;
        ELSE
                ASK OutRec TO GetConsumptionRate (consumer.dailyConsumption);
        END IF;
        consumer := ASK Consumers Next(consumer);
UNTIL(consumer = NILOBJ);
END METHOD {GetConsumptionRates};
```

```
{-------------------------------------------------------}
 ASK METHOD GetMotorpool(IN Motorpool : MotorpoolObj);
{-------------------------------------------------------}
BEGIN
        myMotorpool := Motorpool;
END METHOD {GetMotorpool};

{-------------------------------------------------------}
 ASK METHOD GetSupplyRecord(IN itemName : STRING;
                        IN SupplyClass : STRING;
                        OUT Record : SupplyRecordObj;
                        IN nonseparable : BOOLEAN);
{-------------------------------------------------------}
VAR
        supplyClass : SupplyClassType;
        daysOSupply  : INTEGER;
        stocklvl, upperlvl : REAL;
BEGIN
NEW(Record);
supplyClass := SupplyClass;
stocklvl := 0.0;
daysOSupply := 0;
upperlvl := 0.0;
ASK Record TO GetFields (supplyClass, stocklvl, daysOSupply, upperlvl, nonseparable);
END METHOD {GetSupplyRecord};

{-------------------------------------------------------}
 ASK METHOD GetSupplySource (IN SupplySource : SupplyObj);
{-------------------------------------------------------}
BEGIN
mySupplySource := SupplySource;
END METHOD {GetSupplySource};

{-------------------------------------------------------------}
 TELL METHOD PrepareToStartSupply(IN dayToEndConsuming : INTEGER);
{-------------------------------------------------------------}
VAR
        i : INTEGER;
        ResupplyList : RequestTypeQueue;
        SupplyRec : SupplyRecordObj;
        twentyFourHours : REAL;
BEGIN
twentyFourHours := 24.0;
FOR i := 1 TO dayToEndConsuming
        TELL SELF TO CheckStock IN (twentyFourHours + timeToCheckStock);
        twentyFourHours := twentyFourHours + 24.0;
END FOR;
END METHOD {PrepareToStartSupply};

{-------------------------------------------------------}
 TELL METHOD Resupply (IN RequestList : RequestTypeQueue);
{-------------------------------------------------------}
VAR
        request : RequestObj;
        cargoToLoad : CargoTypeQueue;
BEGIN
WAIT FOR SELF TO CollectItems(RequestList);
        ON INTERRUPT TERMINATE;
END WAIT;
DisposeOfQueue(RequestList);
END METHOD {Resupply};
```

```
{-------------------------------------------------------------}
 TELL METHOD CollectItems (IN RequestList : RequestTypeQueue);
{-------------------------------------------------------------}
VAR
        Request,CloneRequest, AwaitedRequest : RequestObj;
        CargoToLoad, RestToLoad : CargoTypeQueue;
        SupplyRequestList, AwaitingRequestList : RequestTypeQueue;
        OutRec : SupplyRecordObj;
        goodItem : BOOLEAN;
        itemName, Requestor : STRING;
        cargo : CargoObj;
        leftToFill : REAL;
BEGIN
NEW(SupplyRequestList);
NEW(AwaitingRequestList);
NEW(CargoToLoad);
Request := ASK RequestList First();
Requestor := Request.requestor;
REPEAT
        itemName := Request.item;
        FindRec(itemName, supplyRecords, OutRec);
        IF(OutRec = NILOBJ)
                {ASK SELF TO GetSupplyRecord(itemName,0.0,0.0,0.0, FALSE);
                NEW(CloneRequest);
                ASK SupplyRequestList TO Add (CloneRequest);
                CloneRequest := CLONE(Request);
                ASK AwaitingRequestList TO Add (CloneRequest);}
        ELSE
                goodItem := FALSE;
                FindCargo(OutRec, inventory, cargo, Request.amountReq,
                        leftToFill);
                FillRequest(SELF.name, Request.requestor, OutRec, cargo, CargoToLoad, SupplyRequestList, AwaitingRequestList,
                        leftToFill);
        END IF;
        Request := ASK RequestList Next(Request);
UNTIL(Request = NILOBJ);
IF(ASK CargoToLoad numberIn > 0)
        TELL myMotorpool TO ScheduleMission(Requestor, CargoToLoad, NILOBJ);
END IF;
IF(ASK SupplyRequestList numberIn > 0)
        WAIT FOR mySupplySource TO Resupply(SupplyRequestList)
        ON INTERRUPT
        TERMINATE;
        END WAIT;
        NEW(RestToLoad);
        AwaitedRequest := ASK AwaitingRequestList First();
        WHILE(AwaitedRequest < > NILOBJ);
                itemName := AwaitedRequest.item;
                FindRec(itemName, supplyRecords, OutRec);
                IF OutRec = NILOBJ
                        ASK TraceStream TO WriteString("item not found");
                END IF;
                FindCargo(OutRec, inventory, cargo, AwaitedRequest.amountReq,
                                leftToFill);
                IF cargo = NILOBJ
                        ASK TraceStream TO WriteString("should have been cargo");
                ELSE
                        ASK RestToLoad TO Add (cargo);
                END IF;
                AwaitedRequest := ASK AwaitingRequestList Next(AwaitedRequest);
        END WHILE;
```

```
        TELL myMotorpool TO ScheduleMission(Requestor, RestToLoad, NILOBJ);
END IF;
DisposeOfQueue(SupplyRequestList);
DisposeOfQueue(AwaitingRequestList);
DisposeOfQueue(CargoToLoad);
DisposeOfQueue(RestToLoad);
END METHOD {CollectItems};

{-------------------}
 TELL METHOD CheckStock;
{-------------------}
VAR
        Record : SupplyRecordObj;
        RequestList : RequestTypeQueue;
        Request : RequestObj;
        amount : REAL;
BEGIN
NEW(RequestList);
Record := ASK supplyRecords First ();
REPEAT
        IF(Record.stockLevel < = (Record.dailyConsumption *
          FLOAT(Record.daysOfSupply))) AND (Record.upperLevel -
          Record.stockLevel - Record.onOrder > 0.0)
                amount := Record.upperLevel - Record.stockLevel -
                        Record.onOrder;
                NEW(Request);
                ASK Request TO GetFields (Record.supplyClass, amount, SELF.location);
                ASK RequestList TO Add(Request);
        END IF;
        Record := ASK supplyRecords Next(Record);
UNTIL(Record = NILOBJ);
IF(RequestList < > NILOBJ)
        IF(ASK RequestList numberIn > 0)
                TELL mySupplySource TO Resupply(RequestList);
        END IF;
END IF;
DisposeOfQueue(RequestList);
END METHOD {CheckStock};

{-----------------------------------------------------}
 TELL METHOD ReceiveSupplies (IN AssetList : AssetTypeQueue;
                        IN priority : REAL;
                        IN ReceivingPoint : SupplyPointObj);
{-----------------------------------------------------}
VAR
        asset : AssetObj;
        OutRec : SupplyRecordObj;
        item : CargoObj;
        MHEused : INTEGER;
BEGIN
MHEused := 0;
asset := ASK AssetList First ();
REPEAT
        WAIT FOR ReceivingPoint.MHE TO Give (SELF, 1);
                INC(MHEused);
                WHILE (ASK asset.cargoHold numberIn > 0)
                        item := ASK asset.cargoHold TO Remove ();
                        FindRec(item.classOfSupply, supplyRecords, OutRec);
                        IF(OutRec = NILOBJ)
                                GetSupplyRecord(item.classOfSupply, item.classOfSupply, OutRec, TRUE);
                                ASK supplyRecords TO Add(OutRec);
```

```
                        END IF;
                        WAIT FOR asset TO UnloadCargo (0.0, item.weight, item.length);
                        END WAIT;
                        IF(OutRec < > NILOBJ)
                                AdmitItem(OutRec, item);
                        ELSE
ASK TraceStream TO WriteString("Got a NILOBJ for OUTREC! ");
ASK TraceStream TO WriteLn;
                        END IF;
                END WHILE;
                IF(MHEused > = ReceivingPoint.MHE.MaxResources)
                        WAIT DURATION unloadTime
                        END WAIT;
                        ASK ReceivingPoint.MHE TO TakeBack(SELF, MHEused);
                        MHEused := 0;
                END IF;
          END WAIT;
          asset := ASK AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        IF((MHEused < ReceivingPoint.MHE.MaxResources) AND (MHEused < > 0))
                WAIT DURATION unloadTime
                END WAIT;
                ASK ReceivingPoint.MHE TO TakeBack(SELF, MHEused);
        END IF;
END METHOD {ReceiveSupplies};

{-------------------------------------------------------}
 TELL METHOD LoadSupplies (IN AssetList : AssetTypeQueue;
                   IN priority : REAL;
                   IN LoadingPoint : SupplyPointObj);
{-------------------------------------------------------}
VAR
        asset : AssetObj;
        MHEused : INTEGER;
BEGIN
MHEused := 0;
asset := ASK AssetList First ();
REPEAT
        WAIT FOR LoadingPoint.MHE TO Give (SELF, 1);
                INC(MHEused);
                IF(MHEused > = LoadingPoint.MHE.MaxResources)
                        WAIT DURATION loadTime
                        END WAIT;
                        ASK LoadingPoint.MHE TO TakeBack(SELF, MHEused);
                        MHEused := 0;
                END IF;
          END WAIT;
          asset := ASK AssetList Next (asset);
UNTIL (asset = NILOBJ);
IF((MHEused < LoadingPoint.MHE.MaxResources) AND (MHEused < > 0))
        WAIT DURATION loadTime
        END WAIT;
        ASK LoadingPoint.MHE TO TakeBack(SELF, MHEused);
END IF;
END METHOD {LoadSupplies};

{-------------------------------------------------------}
 ASK METHOD AdmitItem (IN SupplyRecord : SupplyRecordObj;
                IN cargo : CargoObj);
{-------------------------------------------------------}
VAR
```

```
        supplyItem : CargoObj;
BEGIN
FindItem(ASK cargo classOfSupply, inventory, supplyItem);
IF(supplyItem = NILOBJ)
        ASK inventory TO Add(cargo);
ELSE
        ASK SELF TO AdjustInventory(supplyItem, cargo.weight);
END IF;
ASK SupplyRecord TO Adjust(cargo.weight, cargo.nonseparable);
DISPOSE(cargo);
END METHOD {AdmitItem};

{-------------------------------------------------------}
 TELL METHOD ReportStatus (IN SupplyReport : StreamObj);
{-------------------------------------------------------}
VAR
        SupplyRec : SupplyRecordObj;
BEGIN
ASK SupplyReport TO WriteString("************************************************************");
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("*******Status for " + name + " SUPPLY POINT.  Time is " + REALTOSTR(SimTime()));
ASK SupplyReport TO WriteLn;

SupplyRec := ASK supplyRecords First ();
REPEAT
        ASK SupplyReport TO WriteString("-------------Record for  " + SupplyRec.supplyClass);
        ASK SupplyReport TO WriteLn;
        ASK SupplyReport TO WriteString("--------------------------");
        ASK SupplyReport TO WriteLn;
        IF SupplyRec.nonSeparable
                ASK SupplyReport TO WriteString("number on hand is  " + INTTOSTR (TRUNC(SupplyRec.stockLevel)));
                ASK SupplyReport TO WriteLn;
                ASK SupplyReport TO WriteString("***************************");
                ASK SupplyReport TO WriteLn;
        ELSE
ASK SupplyReport TO WriteString("stock level is " + REALTOSTR (SupplyRec.stockLevel));
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("days of supply is        " + INTTOSTR (SupplyRec.daysOfSupply));
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("upper level is             " + REALTOSTR (SupplyRec.upperLevel));
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("daily consumption is  " + REALTOSTR (SupplyRec.dailyConsumption));
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("on order is                 " + REALTOSTR (SupplyRec.onOrder));
ASK SupplyReport TO WriteLn;
ASK SupplyReport TO WriteString("***********************************");
ASK SupplyReport TO WriteLn;
END IF;
        SupplyRec := ASK supplyRecords Next (SupplyRec);
UNTIL (SupplyRec = NILOBJ);
END METHOD {ReportStatus};

END OBJECT {Supply};

OBJECT EconomyObj;
{-------------------------------------------------------}
 TELL METHOD Resupply (IN RequestList : RequestTypeQueue);
{-------------------------------------------------------}
VAR
        Request : RequestObj;
        OutRec : SupplyRecordObj;
```

```
        itemName : STRING;
        cargo, separatedCargo : CargoObj;
BEGIN
Request := ASK RequestList First();
REPEAT
        itemName := Request.item;
        FindRec(itemName, supplyRecords, OutRec);
        IF(OutRec = NILOBJ)
                ASK SELF TO GetSupplyRecord(itemName,0.0,0.0,0.0);
        ELSE
                FindItem(itemName, inventory, cargo);
                ASK cargo TO SeparateCargo (Request.amountReq, separatedCargo);
                ASK OutRec TO Adjust (-(Request.amountReq), FALSE);
        END IF;
        FindNode(Request.requestor, Node);
        ASK Node.mySupply TO Admit (separatedCargo);
        Request := ASK RequestList Next(Request);
UNTIL(Request = NILOBJ);

END METHOD {Resupply};

{---------------------}
 TELL METHOD CheckStock;
{---------------------}
VAR
        Record : SupplyRecordObj;
        RequestList : RequestTypeQueue;
        Request : RequestObj;
        amount : REAL;
BEGIN
NEW(RequestList);
Record := ASK supplyRecords First ();
REPEAT
        IF(Record.stockLevel < = (Record.dailyConsumption *
          FLOAT(Record.daysOfSupply))) AND (Record.upperLevel -
          Record.stockLevel - Record.onOrder > 0.0)
                amount := Record.upperLevel - Record.stockLevel -
                        Record.onOrder;
        END IF;
        Record := ASK supplyRecords Next(Record);
UNTIL(Record = NILOBJ);
END METHOD {CheckStock};

END OBJECT {Economy};

END {IMPLEMENTATION} MODULE {supply}.

{*****************************************************}

DEFINITION MODULE SUPREC;
{Supply records for each supply point are defined here}

  FROM GLOBAL IMPORT RequestTypeQueue, SupplyClassType;

TYPE
SupplyRecordObj = OBJECT
        supplyClass : SupplyClassType;      {CL1...CL9, Supply Class}
        stockLevel : REAL;
        daysOfSupply : INTEGER;
        upperLevel : REAL;
        dailyConsumption : REAL;
```

```
        onOrder : REAL;
        nonSeparable : BOOLEAN;

        ASK METHOD GetFields(IN class : SupplyClassType;
                             IN stockL : REAL;
                             IN daysOS : INTEGER;
                             IN upperL : REAL;
                             IN nonseparable : BOOLEAN);
        ASK METHOD GetConsumptionRate(IN ConsumptionRate : REAL);
        ASK METHOD Adjust (IN Adjustment : REAL;
                           IN nonseparable : BOOLEAN);
END OBJECT {SupplyRecord};

END {DEFINITION} MODULE {suprec}.

IMPLEMENTATION MODULE SUPREC;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT SupplyClassType, RequestTypeQueue;
  FROM REQUEST IMPORT RequestObj;

OBJECT SupplyRecordObj;
{-------------------------------------------------}
 ASK METHOD GetFields(IN SupplyClass : SupplyClassType;
                      IN StockLevel : REAL;
                      IN DaysOfSupply : INTEGER;
                      IN UpperLevel : REAL;
                      IN NonSeparable : BOOLEAN);
{-------------------------------------------------}
BEGIN
                      supplyClass := SupplyClass;
                      stockLevel := StockLevel;
                      daysOfSupply := DaysOfSupply;
                      upperLevel := UpperLevel;
                      nonSeparable := NonSeparable;
END METHOD;

{-------------------------------------------------}
 ASK METHOD GetConsumptionRate(IN ConsumptionRate : REAL);
{-------------------------------------------------}
BEGIN
        dailyConsumption := ConsumptionRate;
END METHOD {GetConsumptionRate};

{-----------------------------------------}
 ASK METHOD Adjust(IN Adjustment : REAL;
                   IN nonseparable : BOOLEAN);
{-----------------------------------------}
VAR
        short : REAL;
BEGIN
IF(nonseparable)
        stockLevel := stockLevel + (Adjustment/Adjustment);
ELSE
        stockLevel := stockLevel + Adjustment;
END IF;
END METHOD {Adjust};

END OBJECT {SupplyRecord};

END {IMPLEMENTATION} MODULE {suprec}.
```

```
{************************************************************}

DEFINITION MODULE FINDSUP;
{Contains procedures that necessary in the normal activities of a supply point}
  FROM GLOBAL IMPORT CargoTypeQueue, RequestTypeQueue, SupplyRecordTypeQueue;
  FROM CARGO IMPORT CargoObj;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM REQUEST IMPORT RequestObj;
  FROM RECORDS IMPORT VehiclesRec, VehTypeRec;

{-----------------------------------------------------}
 PROCEDURE FindCargo(IN SupplyRecord : SupplyRecordObj;
                     IN inventory : CargoTypeQueue;
                     OUT cargo : CargoObj;
                     IN amountReq : REAL;
                     OUT leftToFill : REAL);
{-----------------------------------------------------}
{-----------------------------------------------}
PROCEDURE FindItem(IN itemName : STRING;
                   IN inventory : CargoTypeQueue;
                   OUT item : CargoObj);
{-----------------------------------------------}
{-------------------------------------------------------}
 PROCEDURE FillRequest(IN Supply : STRING;
                       IN Requestor : STRING;
                       IN SupplyRecord : SupplyRecordObj;
                       IN cargo : CargoObj;
                       INOUT cargoToMove : CargoTypeQueue;
                       INOUT RequestList : RequestTypeQueue;
                       INOUT AwaitingList : RequestTypeQueue;
                       IN leftToFill : REAL);
{-------------------------------------------------------}
{-------------------------------------------------------}
 PROCEDURE FindRec(IN itemName : STRING;
                   IN SupplyRecords : SupplyRecordTypeQueue;
                   OUT OutRec : SupplyRecordObj);
{-------------------------------------------------------}
{-------------------------------------------------------}
 PROCEDURE FindAssetTypeRecord (IN TypeOfAsset : STRING;
                                IN AssetsRecord : VehiclesRec;
                                OUT AssetTypeRec : VehTypeRec);
{-------------------------------------------------------}

END {DEFINITION} MODULE {findsup}.

IMPLEMENTATION MODULE FINDSUP;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT CargoTypeQueue, RequestTypeQueue, SupplyRecordTypeQueue;
  FROM CARGO IMPORT CargoObj;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM REQUEST IMPORT RequestObj;
  FROM RECORDS IMPORT VehiclesRec, VehTypeRec;

{-----------------------------------------------------}
 PROCEDURE FindCargo(IN SupplyRecord : SupplyRecordObj;
                     IN inventory : CargoTypeQueue;
                     OUT cargo : CargoObj;
                     IN amountReq : REAL;
                     OUT leftToFill : REAL);
{-----------------------------------------------------}
```

```
VAR
        item : CargoObj;
        found : BOOLEAN;
        name : STRING;
        amount : REAL;
BEGIN
item := ASK inventory First ();
found := FALSE;
REPEAT
        IF (item = NILOBJ)
                EXIT;
        END IF;
        name := ASK item classOfSupply;
        IF (SupplyRecord.supplyClass = name)
                found := TRUE;
        ELSE
                item := ASK inventory Next (item);
        END IF;
UNTIL (item = NILOBJ) OR (found);
IF (item = NILOBJ)
        cargo := NILOBJ;
ELSE
        IF (SupplyRecord.stockLevel > amountReq)
                ASK item TO SeparateCargo (amountReq, cargo);
                ASK SupplyRecord TO Adjust (-amountReq, FALSE);
        ELSIF (SupplyRecord.stockLevel > 0.0)
                amount := amountReq - SupplyRecord.stockLevel;
                ASK item TO SeparateCargo (amount, cargo);
                ASK SupplyRecord TO Adjust (-amount, FALSE);
                leftToFill := amountReq - amount;
        ELSE
                cargo := NILOBJ;
                leftToFill := amountReq;
        END IF;
END IF;
END PROCEDURE {FindCargo};

{-----------------------------------------}
PROCEDURE FindItem(IN itemName : STRING;
                IN inventory : CargoTypeQueue;
                OUT item : CargoObj);
{-----------------------------------------}
VAR
        found : BOOLEAN;
BEGIN
item := ASK inventory First ();
found := FALSE;
REPEAT
        IF (item = NILOBJ)
                EXIT;
        END IF;
        IF ((ASK item classOfSupply) = itemName)
                found := TRUE;
        ELSE
                item := ASK inventory Next (item);
        END IF;
UNTIL (item = NILOBJ) OR (found);
END PROCEDURE {FindItem};

{-----------------------------------------------------}
 PROCEDURE FillRequest(IN Supply : STRING;
```

```
                IN Requestor : STRING;
                IN SupplyRecord : SupplyRecordObj;
                IN cargo : CargoObj;
                INOUT cargoToMove : CargoTypeQueue;
                INOUT RequestList : RequestTypeQueue;
                INOUT AwaitingList : RequestTypeQueue;
                IN leftToFill : REAL);
{-------------------------------------------------------------}
VAR
        Request,copyRequest  : RequestObj;
        item : STRING;
        amountReq : REAL;
BEGIN
IF(SupplyRecord = NILOBJ)
        ASK TraceStream TO WriteString("Called Fill Request with a NILOBJ Record");
        ASK TraceStream TO WriteLn;
END IF;
IF(cargo < > NILOBJ)
        ASK cargoToMove TO Add(cargo);
ELSE
        NEW(Request);
        item := SupplyRecord.supplyClass;
        amountReq := leftToFill;
        ASK Request TO GetFields(item,amountReq,Supply);
        ASK RequestList TO Add(Request);
        NEW(copyRequest);
        item := SupplyRecord.supplyClass;
        amountReq := leftToFill;
        ASK copyRequest TO GetFields(item,amountReq,Requestor);
        ASK AwaitingList TO Add(copyRequest);
END IF;
END PROCEDURE {FillRequest};

{-------------------------------------------------------------}
 PROCEDURE FindRec(IN itemName : STRING;
            IN SupplyRecords : SupplyRecordTypeQueue;
            OUT OutRec : SupplyRecordObj);
{-------------------------------------------------------------}
VAR
        name : STRING;
        notFound : BOOLEAN;
        rec : SupplyRecordObj;
BEGIN
rec := ASK SupplyRecords First ();
notFound := TRUE;
REPEAT
        IF (rec < > NILOBJ)
                name := rec.supplyClass;
        END IF;
        IF (name = itemName)
                notFound := FALSE;
        ELSE
                rec := ASK SupplyRecords Next (rec);
        END IF;
UNTIL (rec = NILOBJ) OR (NOT notFound);
OutRec := rec;
END PROCEDURE {FindRec};

{-------------------------------------------------------------}
 PROCEDURE FindAssetTypeRecord (IN TypeOfAsset : STRING;
                        IN AssetsRecord : VehiclesRec;
```

```
                        OUT AssetTypeRec : VehTypeRec);
{-------------------------------------------------------------}
VAR
        foundIt : BOOLEAN;
BEGIN
AssetTypeRec := ASK AssetsRecord.vehTypeRecList First ();
REPEAT
        IF(AssetTypeRec.vehType = TypeOfAsset)
                foundIt := TRUE;
        ELSE
                AssetTypeRec := ASK AssetsRecord.vehTypeRecList Next (AssetTypeRec);
        END IF;
UNTIL((AssetTypeRec = NILOBJ) OR (foundIt));
IF(NOT foundIt)
        ASK TraceStream TO WriteString("****COULD NOT FIND CORRECT ASSET TYPE RECORD!");
        ASK TraceStream TO WriteLn;
END IF;
END PROCEDURE {FindAssetRecord};

END {IMPLEMENTATION} MODULE {findsup}.

{************************************************************}

DEFINITION MODULE REQUEST;
{Supply points that have shortages in their inventories use this object to request a resupply from their supply sources}

TYPE
RequestObj = OBJECT
        item : STRING;
        amountReq : REAL;
        requestor : STRING;      {Name of Node}

        ASK METHOD GetFields (IN itemName : STRING;
                        IN amount : REAL;
                        IN requestor : STRING);
END OBJECT {RequestObj};

END {DEFINITION} MODULE {request}.

IMPLEMENTATION MODULE REQUEST;

OBJECT RequestObj;

{-------------------------------------------}
        ASK METHOD GetFields (IN Item : STRING;
                        IN AmountReq : REAL;
                        IN Requestor : STRING);
{-------------------------------------------}
BEGIN
        item  := Item;
        amountReq := AmountReq;
        requestor := Requestor;
END METHOD {GetFields};

END OBJECT {RequestObj};

END {IMPLEMENTATION} MODULE {request}.

{*********************************************************}

DEFINITION MODULE FUELPT;
```

```
{All fields and methods of a fuelpoint are defined here}
  FROM ResMod IMPORT ResourceObj;
  FROM GLOBAL IMPORT NodeNameType,CargoTypeQueue;

TYPE
FuelPumpQueue = ResourceObj;

FuelpointObj =  OBJECT
  name : NodeNameType;
        fuelLevel : REAL;
        fuelCap : REAL;
        shortageLevel : REAL;
        refuelTime : REAL;
        numberOfPumps : INTEGER;
        FuelPumps : FuelPumpQueue;

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        TELL METHOD PumpFuel (IN FuelUsed : REAL;
                        IN RefuelTime : REAL);
        ASK METHOD GetFields (IN name : STRING);
END OBJECT {Fuelpoint};

END {DEFINITION} MODULE {fuelpt}.

IMPLEMENTATION MODULE FUELPT;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT NodeNameType,CargoTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType,FuelpointSHArray;
  FROM FINDSH IMPORT FindSHRec;

OBJECT FuelpointObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
        NEW(FuelPumps);
END METHOD {ObjInit};

{-------------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{-------------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{-------------------------------------}
 ASK METHOD GetFields (IN name : STRING);
{-------------------------------------}
VAR
        i : INTEGER;
        FuelpointSHRec : SHierRecType;
BEGIN
FindSHRec (FuelpointSHArray, name, FuelpointSHRec);
i := 1;
REPEAT
        fuelLevel := STRTOREAL (FuelpointSHRec.OwnedString[i]);
        INC(i);
        fuelCap := STRTOREAL (FuelpointSHRec.OwnedString[i]);
        INC(i);
        refuelTime := STRTOREAL (FuelpointSHRec.OwnedString[i]);
```

```
        INC(i);
        numberOfPumps := STRTOINT (FuelpointSHRec.OwnedString[i]);
        ASK FuelPumps TO Create(numberOfPumps);
        INC(i);
UNTIL ((i > HIGH(FuelpointSHRec.OwnedString)) OR
        (FuelpointSHRec.OwnedString[i] = "\\"));
END METHOD {GetFields};

{---------------------------------------}
 TELL METHOD PumpFuel (IN FuelUsed : REAL;
                IN RefuelTime : REAL);
{---------------------------------------}
BEGIN
WAIT DURATION RefuelTime
        fuelLevel := fuelLevel - FuelUsed;
END WAIT;
END METHOD {PumpFuel};

END OBJECT {Fuelpoint};

END {IMPLEMENTATION} MODULE {fuelpt}.

{************************************************************}

DEFINITION MODULE UNITS;
{A node's units methods and fields are defined here. Units consume supplies within ITTSS}
  FROM GrpMod IMPORT QueueObj;
  FROM GLOBAL IMPORT NodeNameType, CargoTypeQueue, SupplyClassType, SupplyRecordTypeQueue,
         RequestTypeQueue, ConsumerQueue;
  FROM RGLOBAL IMPORT SHierRecType;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM REQUEST IMPORT RequestObj;
  FROM SUPPLY IMPORT SupplyObj;

EXPORTTYPE
        UnitsObj = OBJECT; FORWARD;
TYPE
ConsumerObj = OBJECT
        name : STRING;
        dailyConsumption : REAL;
        ASK METHOD GetFields(IN name : STRING;
                        IN dailyConsumption : REAL);
END OBJECT {Consumer};

UnitsObj = OBJECT
        name : NodeNameType;
        dayToStartConsuming : INTEGER;
        dayToEndConsuming : INTEGER;
        Consumers : ConsumerQueue;
        inventory : CargoTypeQueue;
        mySupply : SupplyObj;

        ASK METHOD ObjInit;
        ASK METHOD GetName (IN MyName : STRING);
        ASK METHOD GetMySupply(IN Supply : SupplyObj);
        ASK METHOD GetFields (IN Name : STRING);
        TELL METHOD PrepareForConsuming(IN SupplyRecords : SupplyRecordTypeQueue;
                                IN Inventory : CargoTypeQueue);
        TELL METHOD ConsumeSupplies(IN SupplyRecords : SupplyRecordTypeQueue;
                                IN Inventory : CargoTypeQueue);
```

```
END OBJECT {Units};

END {DEFINITION} MODULE {units}.

IMPLEMENTATION MODULE UNITS;
  FROM GLOBAL IMPORT NodeNameType,CargoTypeQueue,SupplyClassType,
        SupplyRecordTypeQueue,RequestTypeQueue;
  FROM RGLOBAL IMPORT SHierRecType,UnitsSHArray;
  FROM FINDSH IMPORT FindSHRec;
  FROM CARGO IMPORT CargoObj;
  FROM FINDSUP IMPORT FindRec, FindCargo, FindItem, FillRequest;
  FROM SUPREC IMPORT SupplyRecordObj;
  FROM DISPREQ IMPORT DisposeRequestTypeQueue;
  FROM REQUEST IMPORT RequestObj;
  FROM Debug IMPORT TraceStream;
  FROM SUPPLY IMPORT SupplyObj;

OBJECT ConsumerObj;
{-----------------------------------------------}
 ASK METHOD GetFields(IN Name : STRING;
                      IN DailyConsumption : REAL);
{-----------------------------------------------}
BEGIN
name := Name;
dailyConsumption := DailyConsumption;
END METHOD {GetFields};

END OBJECT {Consumer};

OBJECT UnitsObj;
{-----------------}
 ASK METHOD ObjInit;
{-----------------}
BEGIN
NEW(Consumers);
NEW(inventory);
END METHOD {ObjInit};

{-------------------------------------}
 ASK METHOD GetName (IN MyName : STRING);
{-------------------------------------}
BEGIN
        name := MyName;
END METHOD {GetName};

{-----------------------------------------------}
 ASK METHOD GetMySupply (IN Supply : SupplyObj);
{-----------------------------------------------}
BEGIN
        mySupply := Supply;
        ASK mySupply TO GetConsumptionRates(Consumers);
END METHOD {GetMySupply};

{-------------------------------------}
 ASK METHOD GetFields(IN Name : STRING);
{-------------------------------------}
VAR
        j : INTEGER;
        UnitsSHRec : SHierRecType;
        consumer : ConsumerObj;
        itemName  : STRING;
```

```
        item : CargoObj;
        DailyConsumption : REAL;
BEGIN
FindSHRec(UnitsSHArray, Name, UnitsSHRec);
j := I;
dayToStartConsuming := STRTOINT(UnitsSHRec.OwnedString[j]);
INC(j);
dayToEndConsuming := STRTOINT(UnitsSHRec.OwnedString[j]);
INC(j);
REPEAT
        NEW(consumer);
        NEW(item);
        itemName := (UnitsSHRec.OwnedString[j]);
        INC(j);
        DailyConsumption := STRTOREAL (UnitsSHRec.OwnedString[j]);
        INC(j);
        ASK consumer TO GetFields(itemName, DailyConsumption);
        ASK item TO GetSupplyFields (itemName, 0.0, 0.0,  "genCgo", 0.0, FALSE);
        ASK Consumers TO Add(consumer);
        ASK inventory TO Add(item);
UNTIL(UnitsSHRec.OwnedString[j] = "EOF");
END METHOD {GetFields};

{-------------------------------------------------------------}
 TELL METHOD PrepareForConsuming(IN SupplyRecords : SupplyRecordTypeQueue;
                          IN Inventory : CargoTypeQueue);
{-------------------------------------------------------------}
VAR
        i : INTEGER;
        ResupplyList : RequestTypeQueue;
        SupplyRec : SupplyRecordObj;
        supplyRecord : SupplyRecordTypeQueue;
        twentyFourHours : REAL;
BEGIN
twentyFourHours := (24.0 * FLOAT(dayToStartConsuming));
FOR i := dayToStartConsuming TO dayToEndConsuming
        TELL SELF TO ConsumeSupplies(SupplyRecords, Inventory) IN twentyFourHours;
        twentyFourHours := twentyFourHours + 24.0;
END FOR;
END METHOD {PrepareForConsuming};

{-------------------------------------------------------------}
 TELL METHOD ConsumeSupplies(IN SupplyRecords : SupplyRecordTypeQueue;
                      IN Inventory : CargoTypeQueue);
{-------------------------------------------------------------}
VAR
        i : INTEGER;
        item : CargoObj;
        ResupplyList : RequestTypeQueue;
        OutRec : SupplyRecordObj;
        consumer : ConsumerObj;
BEGIN
consumer := ASK Consumers First();
REPEAT
        FindRec(consumer.name, SupplyRecords, OutRec);
        ASK OutRec TO Adjust (-(consumer.dailyConsumption), FALSE);
        FindItem(OutRec.supplyClass, Inventory, item);
        ASK item TO Adjust (-(consumer.dailyConsumption));
        consumer := ASK Consumers Next(consumer);
UNTIL(consumer=NILOBJ);
END METHOD {ConsumeSupplies};
```

```
END OBJECT {Units};

END {IMPLEMENTATION} MODULE {units}.

{*************************************************************}

DEFINITION MODULE CHECKAS;
{The sorting algorithms for matching cargo to assets are in the following procedures. Standard sorting techniques are used and cargo
that cannot be loaded is put into a waiting queue}
  FROM GLOBAL IMPORT AssetTypeQueue;
  FROM CARGO IMPORT CargoObj;
  FROM CONVOY IMPORT ConvoyObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM ASSET IMPORT AssetObj;

{-------------------------------------------------------------}
 PROCEDURE FindAssets (INOUT Motorpool : MotorpoolObj;
                       INOUT Convoy : ConvoyObj;
                       IN Load : CargoObj;
                       INOUT CouldNotLoadQueue : AssetTypeQueue);
{-------------------------------------------------------------}
{-------------------------------------------------------------}
 PROCEDURE CheckLoadedAssets (INOUT Motorpool : MotorpoolObj;                    INOUTConvoy:ConvoyObj;
                              INOUT Load : CargoObj;
                              OUT cargoAllLoaded : BOOLEAN);
{-------------------------------------------------------------}

END {DEFINITION} MODULE {checkas}.

IMPLEMENTATION MODULE CHECKAS;
  FROM Debug IMPORT TraceStream;
  FROM GLOBAL IMPORT  AssetTypeQueue, CargoTypeQueue;
  FROM ASSET IMPORT AssetObj;
  FROM CARGO IMPORT CargoObj;
  FROM FINDSHP IMPORT FindShortestPath;
  FROM CONVOY IMPORT ConvoyObj;
  FROM MOTORPL IMPORT MotorpoolObj;
  FROM RECORDS IMPORT VehTypeRec;
  FROM FINDSUP IMPORT FindAssetTypeRecord;

{INPUT A LOAD, FINDS AN ASSET AND LOADS CARGO IN CARGOHOLDS. IF ASSETS ARE PREVIOUSLY LOADED,
WILL CHECK THEM. RETURNS THE ASSET IN A CONVOY}
{-------------------------------------------------------------}
 PROCEDURE FindAssets (INOUT Motorpool : MotorpoolObj;
                       INOUT Convoy : ConvoyObj;
                       IN Load : CargoObj;
                       INOUT CouldNotLoadQueue : AssetTypeQueue);
{-------------------------------------------------------------}
VAR
        cargoWeight, cargoLength,
        maxCanBeLoaded, maxCanBePutOn : REAL;
        asset, biggestAsset, smallestAsset : AssetObj;
        cargoAllLoaded, allCargoWillFit, assetAvailable : BOOLEAN;
        chosenAsset : AssetObj;
        separatedLoad : CargoObj;
        VehTypeRecord : VehTypeRec;
BEGIN
cargoAllLoaded := FALSE;
IF(Convoy < > NILOBJ)
        CheckLoadedAssets(Motorpool, Convoy, Load, cargoAllLoaded);
END IF;
```

```
{all the cargo possible has been loaded on available space on vehicles already in convoy}
REPEAT
        asset := ASK Motorpool.AlgorithmQueue First ();
        REPEAT
                IF(asset.model = "BIG")
                        biggestAsset := asset;
                ELSIF(asset.model = "SMALL")
                        smallestAsset := asset;
                END IF;
                asset := ASK Motorpool.AlgorithmQueue Next(asset);
        UNTIL(asset = NILOBJ);
        cargoWeight := Load.weight;
        cargoLength := Load.length;
        assetAvailable := FALSE;
        asset := ASK Motorpool.AssetList First ();
        IF (asset = NILOBJ)
                assetAvailable := FALSE;
        END IF;
{algorithm finds smallest asset that will hold cargo. If cargo is too big for any of the assets, the largest asset available will be used
and the cargo will be separated into a useable fit}

  WHILE (asset < > NILOBJ) AND (NOT cargoAllLoaded)
        REPEAT
          IF (ASK asset type = ASK Load MOR)
                        IF (cargoWeight <= asset.assetDimens.weight) AND (asset.assetDimens.weight <
smallestAsset.assetDimens.weight)
                                smallestAsset := asset;
                                allCargoWillFit := TRUE;
                                assetAvailable := TRUE;
                        ELSIF (NOT allCargoWillFit) AND (asset.assetDimens.weight > biggestAsset.assetDimens.weight)
                                biggestAsset := asset;
                                assetAvailable := TRUE;
                        END IF;
                  END IF;
                  asset := ASK Motorpool.AssetList Next (asset);
        UNTIL (asset = NILOBJ);
{Found the right asset to load cargo upon}
        IF allCargoWillFit
                chosenAsset := smallestAsset;
        ELSE
                chosenAsset := biggestAsset;
        END IF;
        IF ((NOT cargoAllLoaded) AND (assetAvailable))
                maxCanBeLoaded := chosenAsset.assetDimens.weight;
                maxCanBePutOn := chosenAsset.assetDimens.length;
                IF (cargoWeight <= maxCanBeLoaded)
                  IF (Load.nonseparable) AND (cargoLength <= maxCanBePutOn)
                        ASK Motorpool.AssetList TO RemoveThis (chosenAsset);
                        FindAssetTypeRecord(chosenAsset.vehType, Motorpool.VehiclesRecord, VehTypeRecord);
                        IF(VehTypeRecord = NILOBJ)
                                ASK TraceStream TO WriteString("GOT a NILOBJ from FindAssetTypeRecord****");
                                ASK TraceStream TO WriteLn;
                        ELSE
                                ASK VehTypeRecord TO GetCommitted;
                        END IF;
                        ASK chosenAsset.loadDimens TO UpdateLengthLoad (Load.length);
                        ASK chosenAsset.loadDimens TO UpdateLoadWeight (Load.weight);
                        TELL Motorpool TO MatchCargoToAsset (chosenAsset, Load);
                        ASK Convoy.AssetList TO Add (chosenAsset);
                        cargoAllLoaded := TRUE;
                  ELSIF(NOT Load.nonseparable)
```

```
                    ASK Motorpool.AssetList TO RemoveThis (chosenAsset);
                    FindAssetTypeRecord(chosenAsset.vehType, Motorpool.VehiclesRecord, VehTypeRecord);
                    IF(VehTypeRecord = NILOBJ)
                        ASK TraceStream TO WriteString("GOT a NILOBJ from FindAssetTypeRecord****");
                        ASK TraceStream TO WriteLn;
                    ELSE
                        ASK VehTypeRecord TO GetCommitted;
                    END IF;
                    ASK chosenAsset.loadDimens TO UpdateLoadWeight (Load.weight);
                    TELL Motorpool TO MatchCargoToAsset (chosenAsset, Load);
                    ASK Convoy.AssetList TO Add (chosenAsset); cargoAllLoaded := TRUE;
                END IF;
            ELSIF (NOT Load.nonseparable)
                    ASK Load TO SeparateCargo (maxCanBeLoaded, separatedLoad);
                    ASK Motorpool.AssetList TO RemoveThis (chosenAsset);
                    FindAssetTypeRecord(chosenAsset.vehType, Motorpool.VehiclesRecord, VehTypeRecord);
                    IF(VehTypeRecord = NILOBJ)
                        ASK TraceStream TO WriteString("GOT a NILOBJ from FindAssetTypeRecord****");
                        ASK TraceStream TO WriteLn;
                    ELSE
                        ASK VehTypeRecord TO GetCommitted;
                    END IF;
                    ASK Convoy.AssetList TO Add (chosenAsset);
                    ASK chosenAsset.loadDimens TO UpdateLoadWeight (separatedLoad.weight);
                    TELL Motorpool TO MatchCargoToAsset (chosenAsset, separatedLoad);
                    END IF;
            END IF;
        END WHILE;
UNTIL (cargoAllLoaded) OR (NOT assetAvailable);
IF (NOT cargoAllLoaded) AND (NOT assetAvailable)
        ASK CouldNotLoadQueue TO Add (Load);
END IF;
END PROCEDURE {FindAssets};

{INPUTS A LOAD.  WILL ADD LOAD TO A LOADED ASSET ALREADY IN THE CONVOY IF ROOM IS AVAILABLE.}
{-------------------------------------------------------------}
PROCEDURE CheckLoadedAssets (INOUT Motorpool : MotorpoolObj;                    INOUTConvoy:ConvoyObj;
                    INOUT Load : CargoObj;
                    OUT cargoAllLoaded : BOOLEAN);
{-------------------------------------------------------------}
VAR
        cargoWeight, cargoLength,
        maxCanBeLoaded, maxCanBePutOn : REAL;
        asset : AssetObj;
        allCargoWillFit, assetAvailable : BOOLEAN;
        separatedLoad : CargoObj;
BEGIN
cargoAllLoaded := FALSE;
asset := ASK Convoy.AssetList First ();
IF (asset < > NILOBJ)
  REPEAT
    IF (asset.type = Load.MOR)
        maxCanBeLoaded := asset.assetDimens.weight -
                        asset.loadDimens.weight;
        maxCanBePutOn := asset.assetDimens.length - asset.loadDimens.length;
        cargoWeight := Load.weight;
        cargoLength := Load.length;
        IF cargoWeight < = maxCanBeLoaded
          IF (Load.nonseparable) AND (cargoLength < = maxCanBePutOn)
                ASK asset.loadDimens TO UpdateLoadWeight(Load.weight);
                ASK asset.loadDimens TO UpdateLengthLoad(Load.length);
```

```
            TELL Motorpool TO MatchCargoToAsset (asset, Load);
            cargoAllLoaded := TRUE;
        ELSIF(NOT Load.nonseparable)
            ASK asset.loadDimens TO UpdateLoadWeight(Load.weight);
            TELL Motorpool TO MatchCargoToAsset (asset, Load);
            cargoAllLoaded := TRUE;
        END IF;
      ELSIF (maxCanBeLoaded > 0.0) AND (NOT Load.nonseparable)
            ASK Load TO SeparateCargo (maxCanBeLoaded,
                                       separatedLoad);
            ASK asset.loadDimens TO UpdateLoadWeight (separatedLoad.weight);
            TELL Motorpool TO MatchCargoToAsset (asset, separatedLoad);
      END IF;
    END IF;
    asset := ASK Convoy.AssetList Next (asset);
  UNTIL (cargoAllLoaded) OR (asset = NILOBJ);
END IF;
END PROCEDURE {CheckLoadedAssets};

END {IMPLEMENTATION} MODULE {checkas}.

{*********************************************************}  DEFINITION MODULE CONVOY;

{All actions of a convoy are defined here.  Everthing from loading, travelling and actions upon entering a node are included}
  FROM GLOBAL IMPORT AssetTypeQueue, NodeTypeQueue;
  FROM ROUTE IMPORT LinkObj, RouteObj;
  FROM SUPPLY IMPORT SupplyObj;
  FROM NODE IMPORT NodeObj;

EXPORTTYPE
      ConvoyObj = OBJECT; FORWARD;
TYPE
ConvoyObj = OBJECT
      name : STRING;
      maxAssets,              {max assets allowed in convoy}
      distTweenAssets,        {distance in feet, between assets}
      distBetweenConvoys,     {distance between 2 convoys}
      totalLength : REAL;     {actual length of the convoy}
      arrivalTime : REAL;     {time convoy arrives at mission destination}
      PMCStime : REAL;        {time required for entire convoy to do PMCS}
      allowBreakdowns : BOOLEAN;      {TRUE if wrecker is not avail}
      fixTime : REAL;         {if wrecker or maint is avail, time req to fix asset}
      missionType : STRING;   {either RESUPPLY, RECOVERY, or}
      home : STRING;          {RETURN(fixed asset)}
      priority : REAL;        {priority of highest cargo is given to convoy}
      routeDist : REAL;       {distance in miles of route to destination}
      returnRouteDist : REAL;         {in miles of return trip home}
      fuelNotNeeded : BOOLEAN;        {if TRUE assets do not have to refuel}
      AssetList : AssetTypeQueue;
      DeadlinedPoint : AssetTypeQueue;

      ASK METHOD ObjInit;
      ASK METHOD ObjTerminate;
      ASK METHOD GetName (IN Name : STRING);
      ASK METHOD GetMissionType (IN Mission : STRING;
                        IN Home : STRING;
                        IN distBetweenConvoy : REAL;
                        IN wreckersAvail : BOOLEAN;
                        IN fixTime : REAL);
      ASK METHOD GetDistances (IN routeDistance : REAL;
                       IN returnRouteDistance : REAL);
```

```
        TELL METHOD Travel (IN Destination : STRING;
                            IN Route : RouteObj;
                            IN MasterNodeList : NodeTypeQueue);
        TELL METHOD EnterNode (IN destination : STRING;
                               IN location : STRING;
                               IN MasterNodeList : NodeTypeQueue;
                               IN Route : RouteObj;
                               IN linkDistance : REAL);
        TELL METHOD CheckAssetStatus (IN Route : RouteObj;
                                      IN currentLocation : NodeObj;
                                      IN MasterNodeList : NodeTypeQueue);
        ASK METHOD FindClosestMaintenance (IN Route : RouteObj;
                                           IN currentLocation : STRING;
                                           IN MasterNodeList : NodeTypeQueue;
                                           OUT maintNode : NodeObj);
        TELL METHOD Refuel (IN location : STRING;
                            IN MasterNodeList : NodeTypeQueue);
        TELL METHOD PerformRecovery (IN Node : NodeObj;
                                     IN Origin : STRING);
        TELL METHOD PerformPMCS (IN location : STRING;
                                 IN MasterNodeList : NodeTypeQueue);
        ASK METHOD GetLength (IN distBetweenVeh : REAL);
        TELL METHOD EnterSupply (IN Supply : SupplyObj);
        ASK METHOD NotRefuelDuringMsn;
        TELL METHOD FixAsset;
        TELL METHOD Break (IN breakTime : REAL);
        TELL METHOD CrossLink (IN CrossTime : REAL);
        TELL METHOD StandDown (IN StandDownTime : REAL);

END OBJECT {Convoy};

END {DEFINITION} MODULE {convoy}.

IMPLEMENTATION MODULE CONVOY;
{This procedure does the travelling from one node to another}
  FROM SimMod IMPORT SimTime;
  FROM ResMod IMPORT ResourceObj;
  FROM GLOBAL IMPORT NodeNameType, LinkTypeQueue, AssetTypeQueue, NodeTypeQueue, ALL roadCharact;
  FROM ROUTE IMPORT LinkObj, RouteObj;
  FROM ASSET IMPORT AssetObj, RecoveryObj;
  FROM NODE IMPORT NodeObj, DeadlinePointObj, DeadlinedQueue;
  FROM FINDNOD IMPORT FindNode;
  FROM Debug IMPORT TraceStream;
  FROM SUPPLY IMPORT SupplyObj, SupplyPointObj;
  FROM FINDSUP IMPORT FindAssetTypeRecord;
  FROM RECORDS IMPORT VehTypeRec;

OBJECT ConvoyObj;
{-----------------------------------}
 ASK METHOD GetName(IN Name : STRING);
{-----------------------------------}
BEGIN
name := Name;
END METHOD {GetName};

{------------------}
 ASK METHOD ObjInit;
{------------------}
BEGIN
NEW(AssetList);
END METHOD {ObjInit};
```

```
{----------------------}
 ASK METHOD ObjTerminate;
{----------------------}
BEGIN
DISPOSE(AssetList);
END METHOD {ObjTerminate};

{-----------------------------------------------------}
 ASK METHOD GetMissionType (IN Mission : STRING;
                            IN Home : STRING;
                            IN distance : REAL;
                            IN wreckersAvail : BOOLEAN;
                            IN repairTime : REAL);
{-----------------------------------------------------}
BEGIN
missionType := Mission;
home := Home;
distBetweenConvoys := distance;
IF(wreckersAvail)
        allowBreakdowns := FALSE;
ELSE
        allowBreakdowns := TRUE;
END IF;
fixTime := repairTime;
END METHOD {GetMissionType};

{-----------------------------------------------------}
 ASK METHOD GetDistances (IN RouteDistance : REAL;
                          IN ReturnRouteDistance : REAL);
{-----------------------------------------------------}
BEGIN
routeDist := RouteDistance;
returnRouteDist := ReturnRouteDistance;
END METHOD {GetDistances};

{-----------------------------------------------------}
 TELL METHOD Travel (IN Destination : STRING;
                     IN Route : RouteObj;
                     IN MasterNodeList : NodeTypeQueue);
{-----------------------------------------------------}
VAR
        asset : AssetObj;
        Link : LinkObj;
        checkRoute : RouteObj;
        Node : NodeObj;
        updatedOdometer, updatedFuelLevel : REAL;
        timeElapsed,
        mostTimeElapsed : REAL;
        TravelTime : REAL;
        clearanceDist : REAL;
        clearanceTime : REAL;
        rate : REAL;
BEGIN
NEW(checkRoute);
NEW(Node);
checkRoute := Route;
Link := ASK Route.LinkRoute First ();
REPEAT
        asset := ASK AssetList First ();
        TravelTime := 0.0;
        REPEAT
```

```
                ASK asset TO UpdateGuages (Link);
                timeElapsed := ASK Link distance/ASK asset rateOfTravel
                        [Link.roadSurface, Link.terrain];
                IF (timeElapsed > mostTimeElapsed)
                        TravelTime := timeElapsed;
                        rate := ASK asset rateOfTravel[Link.roadSurface, Link.terrain];
                END IF;
                asset := ASK AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        clearanceDist := (totalLength + distBetweenConvoys)/5280.0;
        clearanceTime := clearanceDist/rate;
        FindNode(Link.origin, MasterNodeList, Node);

        WAIT FOR Node.clearance TO PriorityGive (SELF, 1, SELF.priority);
                WAIT FOR SELF TO CrossLink (clearanceTime)
                        ASK Node.clearance TO TakeBack (SELF, 1);
                END WAIT;
        END WAIT;

        WAIT FOR SELF TO CrossLink (TravelTime - clearanceTime)
        END WAIT;

        WAIT FOR SELF TO EnterNode (Destination, Link.destin, MasterNodeList, checkRoute, Link.distance);
        END WAIT;
        IF((ASK AssetList numberIn) = 0)
                TERMINATE;
        END IF;
        Link := ASK Route.LinkRoute Next (Link);
UNTIL (Link = NILOBJ);
END METHOD {Travel};

{-------------------------------------------------------}
 TELL METHOD EnterNode (IN destination : STRING;
                IN location : STRING;
                IN MasterNodeList : NodeTypeQueue;
                IN Route : RouteObj;
                IN linkDistance : REAL);
{-------------------------------------------------------}
VAR
        Node : NodeObj;
        convoy : ConvoyObj;
        asset : AssetObj;
        Home : NodeObj;
        milesTraveled : REAL;
        NMC : INTEGER;
BEGIN
FindNode (location, MasterNodeList, Node);
FindNode (home, MasterNodeList, Home);
{do not check asset status if back home}
IF(Node.name < > SELF.home)
        WAIT FOR SELF TO CheckAssetStatus (Route, Node, MasterNodeList);
        END WAIT;
END IF;
IF((ASK AssetList numberIn) = 0)
        TERMINATE;
END IF;
IF ((ASK Node name) = destination)
        arrivalTime := SimTime ();
        IF (missionType = "Resupply")
                ASK Home.myMotorpool.Dispatch TO WriteString("CONVOY " +  SELF.name + " arrived at " + Node.name);
                ASK Home.myMotorpool.Dispatch TO WriteLn;
```

```
                ASK Home.myMotorpool.DispatchTO WriteString("  TIME OF  ARRIVAL is " + REALTOSTR (SimTime()));
                ASK Home.myMotorpool.DispatchTO WriteLn;
                ASK Home.myMotorpool.DispatchTO WriteString(".  Number in  convoy is " + INTTOSTR (ASK AssetList
numberIn));
                asset := ASK AssetList First ();
                REPEAT
                        ASK asset TO CheckForBreakdown;
                        IF(NOT asset.missionCapable)
                                NMC := NMC + 1;
                        END IF;
                        asset := ASK AssetList Next (asset);
                UNTIL (asset = NILOBJ);
                TELL SELF TO EnterSupply(Node.mySupply);
                IF(routeDist > Node.maxMilesAllowed)
                        WAIT FOR SELF TO StandDown (Node.standDownTime)
                        END WAIT;
                END IF;
        ELSIF (missionType = "Recovery")
                WAIT FOR SELF TO PerformRecovery(Node, Route.origin);
                END WAIT;
        ELSE {must be a repaired asset, even if not, let in}
                WHILE(ASK AssetList numberIn < > 0)
                        asset := ASK AssetList Remove ();
                        ASK Node.myMotorpool.AssetList TO Add(asset);
                END WHILE;
        END IF;
ELSE
        milesTraveled := linkDistance;
        IF(milesTraveled >= Node.milesBeforeBreak)
                WAIT FOR SELF TO Break (Node.breakTime)
                END WAIT;
        END IF;
END IF;
END METHOD {EnterNode};

{-------------------------------------------------------}
 TELL METHOD CheckAssetStatus (IN Route : RouteObj;
                        IN currentLocation : NodeObj;
                        IN MasterNodeList : NodeTypeQueue);
{-------------------------------------------------------}
VAR
        i, numberOfAssets : INTEGER;
        asset, copy : AssetObj;
        InspectionQueue : AssetTypeQueue;
        Node, maintNode : NodeObj;
        deadlinedAssets : DeadlinePointObj;
        VehTypeRecord : VehTypeRec;
        distHome, distDestin : REAL;
        Link : LinkObj;
        distance : REAL;
        range, Range, maxRange, MaxRange : REAL;
        atHome, foundLocation, fuelpointFound : BOOLEAN;
BEGIN
{check vehicle maintenance status, if non-mission capable, find shortest distance, either back home or at mission destination. Schedule
recovery mission.  asset will be recovered and hauled back.  Cargo will be loaded off at destination or mission rescheduled if taken
back home}

{performing PMCS, If NMC, put in DeadlinedQueue}
NEW(deadlinedAssets);
NEW(InspectionQueue);
atHome := FALSE;
```

```
numberOfAssets := (ASK AssetList numberIn);
WHILE (ASK AssetList numberIn > 0)
        asset := ASK AssetList Remove ();
        ASK asset TO CheckForBreakdown;
        IF(NOT asset.missionCapable)
                FindNode(asset.home, MasterNodeList, Node);
                IF(allowBreakdowns)
                        FindAssetTypeRecord(asset.vehType, Node.myMotorpool.VehiclesRecord, VehTypeRecord);
                        ASK VehTypeRecord TO GetUncommitted;
                        ASK VehTypeRecord TO GetDeadlined;
                        ASK deadlinedAssets.AssetQueue TO Add (asset);
                        ASK Node.myMaintenance.MaintReport TO WriteString ("Asset NMC at " + currentLocation.name);
                        ASK Node.myMaintenance.MaintReport TO WriteString (". Bumper Number is " + asset.bumperNumber);
                        ASK Node.myMaintenance.MaintReport TO WriteLn;
                        ASK Node.myMaintenance.MaintReport TO WriteString ("  Time of breakdown is " +
REALTOSTR(SimTime()));
                        ASK Node.myMaintenance.MaintReport TO WriteLn;
                        ASK Node.myMaintenance.MaintReport TO WriteLn;
                ELSE
                        ASK InspectionQueue TO Add(asset);
                        IF(NOT asset.tempFix)
                                ASK Node.myMaintenance.MaintReport TO WriteString ("Asset NMC at " +
currentLocation.name);
                                ASK Node.myMaintenance.MaintReport TO WriteString (". Bumper Number is " +
asset.bumperNumber);
                                ASK Node.myMaintenance.MaintReport TO WriteLn;
                                ASK Node.myMaintenance.MaintReport TO WriteString ("  Time of breakdown is " +
REALTOSTR(SimTime()));
                                ASK Node.myMaintenance.MaintReport TO WriteLn;
                                ASK Node.myMaintenance.MaintReport TO WriteLn;
                                ASK asset TO FixTemporarily;
                                WAIT FOR SELF TO FixAsset
                                END WAIT;
                        END IF;
                END IF;
        ELSE
                ASK InspectionQueue TO Add (asset);
        END IF;
END WHILE;

IF((ASK deadlinedAssets.AssetQueue numberIn) > 0)
 {getting home of asset}
        asset := ASK deadlinedAssets.AssetQueue First();
        ASK currentLocation.deadlinePointQueue TO Add (deadlinedAssets);  {get name of closest maintenance node}
        FindClosestMaintenance (Route, currentLocation.name, MasterNodeList, maintNode);
 {took out deadlineAsset below}
        ASK deadlinedAssets TO GetRecoverer(maintNode.name);
        TELL maintNode.myMaintenance TO RecoverAssets(currentLocation.name, ASK deadlinedAssets.AssetQueue numberIn);
{Tell home motorpool that asset is deadlined}
        FindNode(asset.home, MasterNodeList, Node);
        FindAssetTypeRecord(asset.vehType, Node.myMotorpool.VehiclesRecord, VehTypeRecord);
        END IF;
END IF;
{return assets to convoy, finished with inspection}
WHILE (ASK InspectionQueue numberIn > 0);
        asset := ASK InspectionQueue TO Remove ();
        ASK AssetList TO Add (asset);
END WHILE;

{check each assets range to next node with a Fuelpoint.  If all assets can make travel with avail fuel, do not refuel.  If at least one
cannot, refuel all assets.  If link distance too long, carry reserve fuel}
```

```
IF((ASK AssetList numberIn) = 0)
        TERMINATE;
END IF;

IF((currentLocation.name < > Route.destin) AND (NOT fuelNotNeeded))
                { dont worry about refuelling if at destination}
        distance := 0.0;
        Link := ASK Route.LinkRoute First ();
                {Finding distance to next Fuelpoint}
        REPEAT
                IF((ASK Link origin = currentLocation.name) OR (foundLocation))
                        foundLocation := TRUE;
                {find next node that has fuel.  at link destin for Fuelpoint availability}
                        distance := distance + Link.distance;
                        FindNode(Link.destin, MasterNodeList, Node);
                        IF(Node.myFuelpoint < > NILOBJ)
                                fuelpointFound :=TRUE;
                        END IF;
                END IF;
                Link := ASK Route.LinkRoute Next (Link);
        UNTIL((Link = NILOBJ) OR (fuelpointFound));
        asset := ASK AssetList First ();
        REPEAT
                range := (asset.fuelCap - asset.fuelLevel) * (asset.fuelConsump);
                maxRange := asset.fuelCap * asset.fuelConsump;
                IF(range < Range)
                        Range := range;
                END IF;
{After refueling, need to see if distance is still too great for fully fueled assets}
                IF(maxRange < MaxRange)
                        MaxRange := maxRange;
                END IF;
                asset := ASK AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        IF(Range < distance)
                WAIT FOR SELF TO Refuel (Node.name, MasterNodeList);
                END WAIT;
        END IF;
        IF(MaxRange < distance)
        {carry reserve to next node. have to check if node has a
  fuelpoint, if not, then must carry enough fuel to make it there
  w/delays}
        END IF;
END IF;
END METHOD {CheckAssetStatus};

{-------------------------------------------------------------}
 ASK METHOD FindClosestMaintenance (IN Route : RouteObj;
                            IN currentLocation : STRING;
                            IN MasterNodeList : NodeTypeQueue;
                            OUT maintNode : NodeObj);
{-------------------------------------------------------------}
VAR
        Node : NodeObj;
        distHome, distDestin : REAL;
        Link : LinkObj;
        distance : REAL;
        foundLocation, fuelpointFound : BOOLEAN;
BEGIN
distHome := 0.0;
distDestin := 0.0;
```

```
foundLocation := FALSE;
Link := ASK Route.LinkRoute First ();
REPEAT
        IF((ASK Link origin = currentLocation) OR (foundLocation))
                foundLocation := TRUE;
                distDestin := distDestin + Link.distance;
        ELSE
                distHome := distHome + Link.distance;
        END IF;
        Link := ASK Route.LinkRoute Next (Link);
UNTIL(Link = NILOBJ);
FindNode(Route.destin, MasterNodeList, Node);
IF((distDestin >= distHome) AND (Node.myMaintenance < > NILOBJ))
        maintNode := Node;
ELSE
        FindNode(home, MasterNodeList, Node);
        maintNode := Node;
END IF;
END METHOD {FindClosestMaintenance};

{-------------------------------------------------}
 TELL METHOD Refuel (IN location : STRING;
                IN MasterNodeList : NodeTypeQueue);
{-------------------------------------------------}
VAR
        fuelUsed : REAL;
        asset : AssetObj;
        Node : NodeObj;
        pumpsUsed : INTEGER;
BEGIN
pumpsUsed := 0;
FindNode(location, MasterNodeList, Node);
IF (Node.myFuelpoint < > NILOBJ)
        asset := ASK AssetList First ();
        IF(asset.fuelLevel < > asset.fuelCap)
         REPEAT
          WAIT FOR Node.myFuelpoint.FuelPumpsTO Give (SELF, 1);
                INC(pumpsUsed);
                WAIT FOR Node.myFuelpointTO PumpFuel (fuelUsed, 0.0);
                        ASK Node.myFuelpoint.FuelPumpsTO TakeBack (SELF, 1);
                        ASK asset TO Refuel (fuelUsed);
                        IF(pumpsUsed >= Node.myFuelpoint.FuelPumps.MaxResources)
                                WAIT DURATION Node.myFuelpoint.refuelTime
                                END WAIT;
                                pumpsUsed := 0;
                        END IF;
                END WAIT;
          END WAIT;
        asset := ASK AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        IF((pumpsUsed < Node.myFuelpoint.FuelPumps.MaxResources)OR  (pumpsUsed < > 0))
                        WAIT DURATION Node.myFuelpoint.refuelTime
                        END WAIT;
        END IF;
 END IF;
END IF;
END METHOD {Refuel};

{-----------------------------------------------}
 TELL METHOD PerformRecovery (IN Node : NodeObj;
                        IN Origin : STRING);
```

```
{-------------------------------------------}
VAR
        deadLinedAssets : DeadlinePointObj;
        wrecker : RecoveryObj;
        brokenAsset : AssetObj;
        foundDeadlinedAssets : BOOLEAN;
BEGIN
foundDeadlinedAssets := FALSE;
deadLinedAssets := ASK Node.deadlinePointQueue First ();
REPEAT
        IF (deadLinedAssets.recoverer = Origin)
                foundDeadlinedAssets := TRUE;
        ELSE
                deadLinedAssets := ASK Node.deadlinePointQueue Next (deadLinedAssets);
        END IF;
UNTIL ((deadLinedAssets = NILOBJ) OR (foundDeadlinedAssets));
wrecker := ASK AssetList First ();
REPEAT
        IF((ASK deadLinedAssets.AssetQueue numberIn) > 0)
                brokenAsset := ASK deadLinedAssets.AssetQueue Remove();
                WAIT FOR wrecker TO HookUp(brokenAsset);
                END WAIT;
        ELSE
                ASK TraceStream TO WriteString("could not find deadlined asset");
                ASK TraceStream TO WriteLn;
        END IF;
        wrecker := ASK AssetList Next (wrecker);
UNTIL(wrecker = NILOBJ);
END METHOD {PerformRecovery};

{-----------------------------------------------------------}
 TELL METHOD PerformPMCS (IN location : STRING;
                  IN MasterNodeList : NodeTypeQueue);
{-----------------------------------------------------------}
VAR
        fuelUsed : REAL;
        i, numberOfAssets, pumpsUsed : INTEGER;
        asset : AssetObj;
        DeadlinedQueue, InspectionQueue : AssetTypeQueue;
        Node : NodeObj;
        foundLocation, fuelpointFound : BOOLEAN;
        VehTypeRecord : VehTypeRec;
BEGIN
{performing PMCS, If NMC, put in DeadlinedQueue}
NEW(DeadlinedQueue);
NEW(InspectionQueue);
FindNode(location, MasterNodeList, Node);
numberOfAssets := (ASK AssetList numberIn);

WAIT DURATION PMCStime
END WAIT;
FOR i := 1 TO numberOfAssets
        asset := ASK AssetList Remove ();
        IF (asset < > NILOBJ)
                ASK asset TO CheckForBreakdown;
                IF(NOT asset.missionCapable)
                        ASK DeadlinedQueue TO Add (asset);
                ELSE
                        ASK InspectionQueue TO Add (asset);
                END IF;
        END IF;
```

```
END FOR;
IF((ASK DeadlinedQueue numberIn) > 0)
        numberOfAssets := (ASK DeadlinedQueue numberIn);
        FOR i := 1 TO numberOfAssets
                asset := ASK DeadlinedQueue Remove();
                FindAssetTypeRecord(asset.vehType, Node.myMotorpool.VehiclesRecord, VehTypeRecord);
                IF(VehTypeRecord = NILOBJ)
                        ASK TraceStream TO WriteString("GOT a NILOBJ from FindAssetTypeRecord****");
                        ASK TraceStream TO WriteLn;
                ELSE
                        ASK VehTypeRecord TO GetDeadlined;
                        ASK VehTypeRecord TO GetUncommitted;
                END IF;
                TELL Node.myMaintenance TO ReceiveWork(asset);
        END FOR;
END IF;
numberOfAssets := (ASK InspectionQueue numberIn);
IF(numberOfAssets > 0)
        FOR i := 1 TO numberOfAssets
                asset := ASK InspectionQueue TO Remove ();
                IF(asset < > NILOBJ)
                        ASK AssetList TO Add (asset);
                END IF;
        END FOR;
END IF;
{Refuel Assets}
IF (Node.myFuelpoint < > NILOBJ)
        pumpsUsed := 0;
        asset := ASK AssetList First ();
        REPEAT
                ASK asset TO ResetTripOdometer;
                WAIT FOR Node.myFuelpoint.FuelPumpsTO Give (SELF, 1);
                  WAIT FOR Node.myFuelpoint TO PumpFuel (fuelUsed, 0.0);
                        ASK Node.myFuelpoint.FuelPumpsTO TakeBack (SELF, 1);
                        ASK asset TO Refuel (fuelUsed);
                        IF(pumpsUsed > = Node.myFuelpoint.FuelPumps.MaxResources)
                                WAIT DURATION Node.myFuelpoint.refuelTime
                                END WAIT;
                                pumpsUsed := 0;
                        END IF;
                  END WAIT;
                END WAIT;
                asset := ASK AssetList Next (asset);
        UNTIL (asset = NILOBJ);
        IF((pumpsUsed < Node.myFuelpoint.FuelPumps.MaxResources)OR  (pumpsUsed < > 0))
                        WAIT DURATION Node.myFuelpoint.refuelTime
                        END WAIT;
        END IF;
END IF;
END METHOD {PerformPMCS};

{-----------------------------------------------}
 ASK METHOD GetLength (IN distBetweenVeh : REAL);
{-----------------------------------------------}
VAR
        asset : AssetObj;
BEGIN
totalLength := 0.0;
asset := ASK AssetList First ();
totalLength := asset.assetLength;
REPEAT
```

```
        asset := ASK AssetList Next(asset);
        IF(asset < > NILOBJ)
                totalLength := totalLength + asset.assetLength +  distBetweenVeh;
        END IF;
UNTIL(asset = NILOBJ);
END METHOD {GetLength};

{-----------------------------------------------}
 TELL METHOD EnterSupply (IN Supply : SupplyObj);
{-----------------------------------------------}
VAR
        ReceivingPt : SupplyPointObj;
BEGIN
WAIT FOR Supply.receivingPointsQueueTO PriorityGive (SELF, 1,  SELF.priority);
        ReceivingPt := ASK Supply.ReceivingPoints TO Remove();
        WAIT FOR Supply TO ReceiveSupplies(AssetList, SELF.priority,  ReceivingPt);
                ASK Supply.receivingPointsQueueTO TakeBack (SELF, 1);
                ASK Supply.ReceivingPoints TO Add(ReceivingPt);
        END WAIT;
END WAIT;
END METHOD {EnterSupply};

{-----------------------------}
 ASK METHOD NotRefuelDuringMsn;
{-----------------------------}
BEGIN
        fuelNotNeeded := TRUE;
END METHOD {NotRefuelDuringMsn};

{--------------------}
 TELL METHOD FixAsset;
{--------------------}
BEGIN
        WAIT DURATION fixTime
        END WAIT;
END METHOD {FixAsset};

{---------------------------------------}
 TELL METHOD Break (IN breakTime : REAL);
{---------------------------------------}
BEGIN
        WAIT DURATION breakTime
        END WAIT;
END METHOD {Break};

{-----------------------------------------}
 TELL METHOD CrossLink (IN CrossTime : REAL);
{-----------------------------------------}
BEGIN
        WAIT DURATION CrossTime
        END WAIT;
END METHOD {CrossLink};
```

```
{-----------------------------------------------}
 TELL METHOD StandDown (IN StandDownTime : REAL);
{-----------------------------------------------}
BEGIN
        WAIT DURATION StandDownTime
        END WAIT;
END METHOD {StandDown};

END OBJECT {Convoy};

END {IMPLEMENTATION} MODULE {convoy}.
```

INITIAL DISTRIBUTION LIST

	copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. LTC William Caldwell Department of Operations Research Naval Postgraduate School, Code OR/CW Monterey, CA 93943-5000	1
4. Professor Michael P. Bailey Department of Operations Research Naval Postgraduate School, Code OR/BA Monterey, CA 93943-5000	1
5. Director Military Traffic Management Command Transportation Engineering Agency 720 Thimble Shoals Boulevard Newport News, VA 23606-2574	1
6. Commander and Director U.S. Army Engineers Waterways Experiment Station ATTN: (CEWES-GM-L (Dr. David Horner) 3909 Halls Ferry Road Vicksburg, MS 39180-6199	1
7. Commander U.S. Army Materiel Systems Analysis Activity ATTN: Mr. Russell Farrell Aberdeen Proving Grounds, MD 21005-5071	1

8. CPT James M. Judy
15 Warren Drive
Newport News, VA 23602 2